NeuADC: Neural Network-Inspired Synthesizable Analog-to-Digital Conversion

Weidong Cao, *Student Member, IEEE*, Xin He, *Member, IEEE*, Ayan Chakrabarti, *Member, IEEE*, and Xuan Zhang, *Member, IEEE*

***Abstract*—Traditional analog-to-digital converters (ADCs) employ dedicated analog and mixed-signal (AMS) circuits, requiring time-consuming manual design process. They also exhibit limited configurability to support diverse quantization schemes on the same circuitry. In this paper, we propose NeuADC—an automated design approach to synthesizing an analog-to-digital (A/D) interface that can approximate the desirable quantization function using a neural network (NN) with a single hidden layer. We leverage the mixed-signal resistive random-access memory (RRAM) crossbar architecture to design a novel dual-path configuration for the implementation of the basic NN operations at the circuit level. We exploit alternative bits encoding scheme to the conventional binary encoding to improve the training accuracy. Our method incorporates non-idealities at the device and circuit level into the training process to ensure NeuADC's robustness against variations of process, supply voltage, and temperature (PVT). Results obtained from SPICE simulation based on RRAM and standard 130-nm CMOS technology suggest that not only can NeuADC deliver promising performance compared to the state-of-the-art ADCs and other emerging converter designs across comprehensive design metrics, but it can also intrinsically support multiple configurable quantization schemes using the same hardware substrate, paving ways for future adaptable application-driven signal conversion. Our systematic evaluations on the proposed NeuADC framework also quantify the impacts on the ADC quantization quality from hidden neuron sizes, RRAM resistance imprecision, and PVT variations, and reveal the design tradeoff between speed, power, and area in a NeuADC circuit.**

***Index Terms*—Analog-to-digital (A/D) conversion, mixed-signal computing, multiple quantization schemes, neural network (NN), resistive random-access memory (RRAM) crossbar, synthesizable.**

Manuscript received December 17, 2018; revised March 4, 2019 and April 29, 2019; accepted June 3, 2019. Date of publication June 27, 2019; date of current version August 20, 2020. This work was supported in part by the National Science Foundation under Grant CNS-1657562. This paper was recommended by Associate Editor X. Bonani. *(Corresponding author: Weidong Cao.)*

W. Cao and X. Zhang are with the Department of Electrical and Systems Engineering, Washington University in St. Louis, St. Louis, MO 63130 USA (e-mail: weidong.cao@wustl.edu; xuan.zhang@wustl.edu).

X. He was with the Department of Electrical and Systems Engineering, Washington University in St. Louis, St. Louis, MO 63130 USA. He is now with the University of Michigan, Ann Arbor, MI USA (e-mail: hex0102@gmail.com).

A. Chakrabarti is with the Department of Computer Science and Engineering, Washington University in St. Louis, St. Louis, MO 63130 USA (e-mail: ayan@wustl.edu).

Digital Object Identifier 10.1109/TCAD.2019.2925391

I. Introduction

As INTEGRATED circuits scale to more advanced technology with lower supply voltage, many challenges arise in designing scalable analog and mixed-signal (AMS) circuits [1]–[5], such as reduced intrinsic device gain, decreased signal swing, and aggravated device mismatch. As one of the quintessential examples of AMS circuits, analog-to-digital converters (ADCs) face similar design challenges when being ported to smaller highly scaled technology nodes [5], [6]. Traditional ADC circuits often require significant manual design iterations and respins to meet the desirable performance specifications in a new process. Previous research has explored synthesizable and scaling-compatible ADC topologies to automate this expensive and time-consuming design process [5]–[10]. One example is the stochastic flash ADCs that make use of the intrinsic input offsets of minimum-sized digital comparators [5], [6]. However, stochastic ADCs require a large number of hardware resources (∼3840 comparators) and work only at relatively modest sampling rate (∼8 MS/s) and resolution (∼5.3 bits). Another example is synthesis-friendly time-domain delta-sigma ADCs [7], but they still require manual modifications of a standard cell and designer's knowledge for floor-planning. Other automated strategies are also proposed to design successive approximation (SAR) ADCs at the sub-block level, instead of a holistic automation approach for the full system [8]–[10]. Despite its crucial role, the lack of effective design automation severely limits the productivity improvement and performance enhancement for the AMS circuits, causing major bottleneck in IC development.

In addition to the design automation challenge, ADCs also face new demands from many emerging applications [11]–[19]. For example, in-memory computation (IMC) using nonvolatile memory (NVM) crossbar arrays has been proposed for deep learning applications, where ADCs play a critical role in converting the computed analog signal on the bitline (BL) of the NVM crossbar array to digital bits for further processing [13]–[15]. Due to the specific BL computation mechanism, nonuniform quantization and adaptive tuning of the quantization threshold are often required, which is difficult to achieve by the conventional ADC design with fixed quantizations and thresholds. Therefore, an ADC design that intrinsically supports multiple quantization schemes to satisfy the specifications of the IMC systems is highly desirable. The ability to support flexible quantization schemes is also a desirable property that can benefit a variety of sensor front-end interfaces where ADCs reside in [11] and [12]. For instance, image sensors require logarithmic quantizations to realize

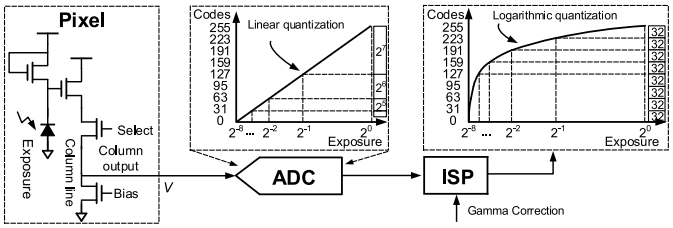


Fig. 1. Standard ADC in image sensor with linear quantization.

uniform distribution of exposure [11], [12]. Traditionally, as shown in Fig. 1, the logarithmic quantization step is performed in the digital domain by the back-end image signal processor (ISP) through a specific gamma correction stage, since the standard ADCs used in the image sensor front-end only implement normal linear quantization. Therefore, a reconfigurable front-end ADC supporting different quantization schemes can obviate the need to perform certain image processing steps, such as gamma correction later in the digital domain, enabling better preservation of useful information, and improved power saving.

In this paper, we propose NeuADC—a novel design approach for automatic ADC synthesis that can address the aforementioned imminent challenges facing the traditional ADC design paradigm [16]. We term our new design NeuADC, as it is inspired by neural network (NN). Our approach is founded on a deep learning framework and implemented by using mixed-signal resistive random-access memory (RRAM) crossbar architecture. We consider RRAM, a promising NVM technology [20], a perfect testbed to demonstrate the scaling-compatible and portability-friendly features of our method. Essentially, the proposed NeuADC framework formulates the ADC design as an NN learning problem with the learning objective of approximating multiple desirable quantization functions for A/D conversion. This allows us to take advantages of many effective training techniques developed for deep learning and seamlessly incorporate them into ADC design automation. The following key innovations and contributions are made in this paper.

1) We propose an NN-inspired design methodology to model the A/D interfaces and transform the traditional ADC design problem into a learning problem. Our novel design approach opens new opportunities to employ learning techniques in AMS design automation.
2) We propose a new dual-path RRAM crossbar architecture to facilitate mixed-signal vector matrix multiplication (VMM) required by NeuADC in a scaling-compatible manner, along with an inverter voltage transfer curve (VTC)-based activation function to implement the nonlinear activation function (NAF).
3) We explore offline training techniques and a smooth-bit encoding scheme to obtain robust trained weight parameters that account for device and circuit level nonidealities, including process, supply voltage, and temperature (PVT) variations of the CMOS transistors and the limited resolution of the RRAM devices.
4) We develop a fully automated design flow to synthesize the proposed NeuADC circuits and present SPICE simulation results based on the automatically synthesized netlist. Our SPICE simulation results validate the competitive performance of the proposed NeuADC and its ability to support multiple quantization schemes, and also reveal the impacts on the ADC quantization quality from the NN-level parameters (i.e., hidden neuron sizes and the number of output bits) and device-level characteristics, as well as the design tradeoff between speed, power and area in a NeuADC circuit.

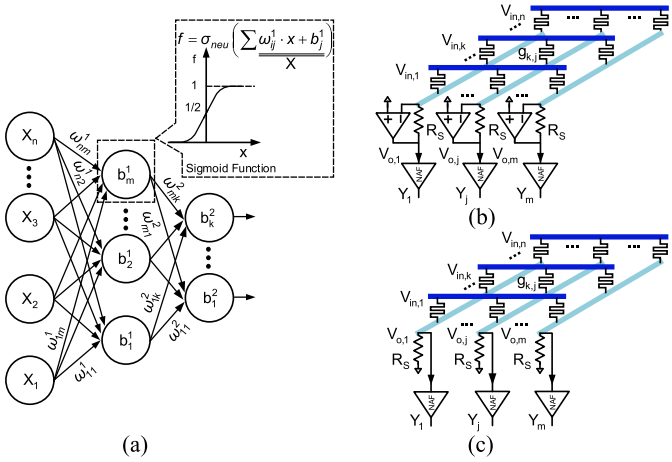


Fig. 2. (a) Illustration of three-layer NN universal approximator. (b) RRAM crossbar array with Op-Amps. (c) RRAM crossbar array without Op-Amps.

The rest of this paper is organized as follows. Section II provides the preliminary background and related work on this topic. An overview of the proposed NeuADC design methodology is summarized in Section III, and more detailed implementations of the hardware substrate and the training framework are described in Sections IV and V, respectively. Finally, we present the simulation methodology and results in Sections VI and VII before concluding this paper in Section VIII.

II. Background and Related Work

To provide the background context for this paper, we first briefly introduce the basic multilayer perceptron (MLP) model with universal approximation property and then give a quick overview of the RRAM technology and its crossbar architecture, which is often used to perform efficient VMM step of the NN operations. Finally, we review the related work that has explored the possibility of using NN-inspired principles to realize analog-to-digital (A/D) conversion.

A. Multilayer Perceptron

An MLP is a class of feedforward artificial NN (ANN) which consists of, at least, three layers of nodes: 1) an input layer; 2) a hidden layer; and 3) an output layer. Except for the input nodes, each node is a neuron that uses an NAF. A simple model of a three-layer feedforward MLP with one hidden layer is illustrated in Fig. 2(a). The basic NN operations between neighbor layers can be expressed as

$$y_j = \sigma_j\left(\sum_{i=1}^{n}\left(w_{ij}\cdot x_i + b_j\right)\right) \tag{1}$$

where x_i is the input signal value of node $i, i = 1, 2, \ldots, n$ in the input layer, and y_j is the output signal of node $j, j = 1, 2, \ldots, m$ in the hidden layer. w_{ij} is the weight to connect input x_i and output y_j. b_j is the offset for jth neuron in the

hidden layer. $\sigma_j(x)$ is the NAF, e.g., a sigmoid function

$$\sigma_j(x) = \frac{1}{1+e^{-x}}. \tag{2}$$

It has been proven that arbitrary decision regions can be arbitrarily well approximated by continuous feedforward NN with a single hidden layer and any continuous sigmoid activation function [33]. Therefore, if these operations can be implemented on hardware NN, it is possible to train such NN to approximate an ADC with different quantization schemes.

B. RRAM Technology for NN Computing

Although originally developed as an NVM technology, the emerging RRAM has been proposed as a promising technology for implementing NN approximate computing engines and learning accelerators, owing to its crossbar architecture enabling efficient in-memory matrix multiplication [14], [15]. RRAM is a passive two-terminal device built on TiOx, WOx, HfOx,[1] or other materials with variable resistance properties [20], and has special advantages in small cell size ($4F^2$, F is the minimum feature size), excellent scalability (<10 nm) [20], fast switching time (<10 ns) [20], [27], and good endurance (up to 10^{12} cycles) [20]. Another advantage of RRAM is the CMOS-compatible fabrication process and monolithic 3-D integration [20]. Therefore, RRAM cells can be organized into stackable and ultradense crossbar arrays at no extra area overhead. As the focus of this paper is not on technology, we use the SPICE model of the RRAM to simulate, and analyze the performance of circuits [24]–[26].

RRAM crossbar array-based NN computation consists of two essential functions at its heart: 1) a VMM (1), to associate network weights with output from previous layer and 2) an NAF (2), to convert the summation of current layer to the input of next layer. Fig. 2(b) and (c) illustrates two common RRAM crossbar arrays to implement VMM on hardware [15], and the VMM computation between two adjacent layers are expressed as

$$V_{o,j} = \sum_{i=1}^{n} c_{ij} \cdot V_{\text{in},i} \tag{3}$$

where $V_{\text{in},i}$ is the signal value of input node i, $i = 1, 2, \ldots, n$, and $V_{o,j}$ is the signal of output node j, $j = 1, 2, \ldots, m$; c_{ij} is the weight to connect input $V_{\text{in},i}$ and output $V_{o,j}$. In terms of the crossbar in Fig. 2(b) with operational amplifiers (Op-Amps), c_{ij} is linear with the g_{ij} and can be expressed as

$$c_{ij} = -\frac{g_{ij}}{g_s}. \tag{4}$$

While for the crossbar array in Fig. 2(c) without Op-Amps, the c_{ij} can be expressed as

$$c_{ij} = \frac{g_{ij}}{g_s + \sum_{k=1}^{n} g_{ik}} \tag{5}$$

where g_s is the conductance of summation resistor R_s and the g_{ij} is the conductance of memristor in the ith row and jth column of the crossbar array. After VMM, the output $V_{o,j}$ will be fed into an NAF, which usually is realized as piecewise look-up table (LUT) or customized quasi-sigmoid function [15], [28], to generate the input for next layer.

[1]In this paper, we choose HfOx-based RRAM device as the core component of crossbar array, since it is one of the most mature materials which have been explored so far.

Both types of the RRAM crossbar arrays are efficient to realize VMM by reducing the computation complexity from $O(n^2)$ to $O(1)$. However, they cannot be used in our design because Op-Amps and resistors are not suitable for synthesizable design style. In Section IV, we will illustrate a new architecture of RRAM crossbar and CMOS inverter-based neuron to implement the hardware substrate of NeuADC.

C. Related Work

Previous work has explored applying NN principles in ADC architectures [21]–[23]. For example, 4-bit ADCs based on either Hopfield NN or T-model NN have been reported before [22], [23]. They are built on a recurrent architecture where resistors array are used to implement basic multiplication operations and inverting amplifiers act as neurons. These basic proof-of-concept designs are implemented on breadboards by fine-tuning the discrete components to show a sampling rate on the order of 10 S/s. Another example is a neuromorphic ADC that uses integrate-and-fire neurons as time or rate encoders of the analog input [21]. The resolution of the encoding, in both time and amplitude, is increased by using multiple neurons in parallel. The circuit model of integrate-and-fire neuron is complicated, which restricts the speed of ADC. It is also demonstrated on breadboard using off-the-shelf ICs and discrete components, which is not optimal for power efficiency or device configuration.

These preliminary work often lacks the specific considerations of large-scale highly integrated circuit implementation, impact of technology scaling, customization of the training process, or design automation capabilities. Moreover, their methods do not exploit the explicit formulation of the A/D conversion as a learning problem, and hence is fundamentally different from our proposed work.

III. NeuADC Design Methodology

In this section, we provide an overview of design methodology used in the learning and synthesis of the proposed NeuADC. In order to build an intuitive understanding of our novel approach, we first illustrate how an ADC design can be conceptually mapped to an NN model. We then introduce the basic steps in the design process to facilitate a fully automated ADC design flow.

A. Mapping ADCs to NNs

The mapping methodology is inspired by NN's ability to closely approximate general nonlinear functions. From a mathematical modeling perspective, an ideal ADC converts an analog-valued input V_{in} into a digital bit-vector output according to the uniform staircase quantization function and the analytical expression for each bit can be obtained as a highly nonlinear function. For example, a typical flash ADC shown in Fig. 3(a) represents the straightforward circuit realization of the quantization using ideal comparators and a thermometer-to-binary decoder. Coincidentally, the universal approximation theorem proves that a feed-forward NN with a single hidden layer, also known as MLP, can approximate arbitrary complex functions, given sufficient number of hidden neurons [33].

Based on their respective structures, a straightforward architectural mapping between a flash ADC in Fig. 3(a) and an MLP in Fig. 3(c) can be intuitively obtained. A flash ADC

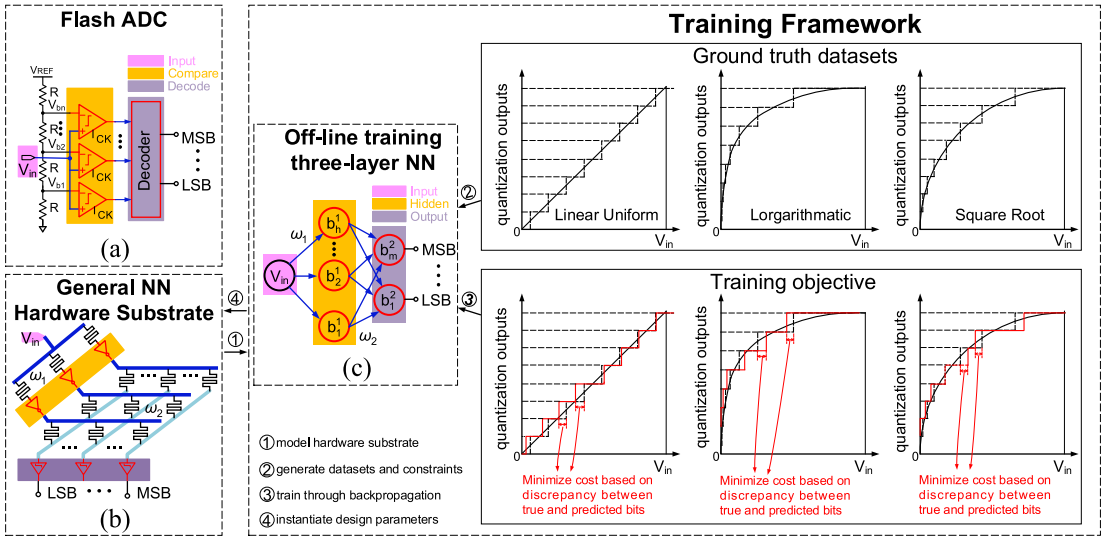


Fig. 3. Conceptual illustration of the proposed NeuADC design methodology. (a) Conventional flash ADC architecture. (b) Proposed NeuADC hardware substrate based on RRAM crossbar. (c) Proposed automated design flow that takes ideal quantization datasets as inputs during offline training to find the optimal set of weights and derive the RRAM resistances in order to minimize the cost function and best approximate the ideal quantization function.

typically consists of three stages—the first input stage takes analog signal from the previous sampling and holding circuit (S/H) [35], [36]; the second compare stage employs a linear resistive voltage ladder to set trip points V_{bi} for a string of comparators; and the last decode stage converts the intermediate thermometer code into binary code. Structurally, the "input-compare-decode" three-stage architecture in the flash ADC resembles the "input-hidden-output" three-layer MLP. Therefore, we conjecture that if a general NN hardware substrate can be implemented, it is possible to train its weights parameters offline to approximate the ideal ADC quantization function.

B. Design Methodology Overview

Fig. 3 provides the conceptual overview of our design methodology founded on the aforementioned mapping between ADCs and NNs. The proposed NeuADC consists of two integrated elements—a general NN hardware substrate and a hardware-oriented training framework, as illustrated in Fig. 3(b) and (c), respectively. We choose RRAM crossbar array and inverter as the hardware substrate to perform general NN operations, such as VMM and NAF. This hardware substrate operates in the mixed-signal domain to map the input analog signal to the output digital bits, which is different from existing NN accelerators [14], [15] that deal exclusively with digital inputs and outputs. The offline training framework can learn the appropriate design parameters for the NN hardware substrate to approximate the specific quantization behavior of an ADC. The overall design process can be summarized in four steps.

1) The behavior of the RRAM crossbar array-based hardware substrate in Fig. 3(b) is modeled as an MLP, as indicated in Fig. 3(c).
2) The training datasets based on the desirable ideal quantization function are fed to the optimization algorithm, along with customized objective functions and constraints to accurately reflect the hardware characteristics of the underlying circuits.
3) The weights of the NN are iteratively trained through back-propagating the output errors.
4) The offline-trained weights are used to instantiate the corresponding design parameters in the hardware substrate.

In this paper, we show successful training for three different quantization schemes (linear uniform encoding, logarithmic encoding, and square root encoding) using the same offline training framework. For each encoding scheme, a group of corresponding offline-trained weights can be obtained. These weights are then used to configure different RRAM resistance values to realize multiple signal quantization schemes on the same NeuADC hardware substrate. We will delve into the design of the hardware substrate and the development of the training framework later sections.

IV. Hardware Substrate

In this section, we describe the circuit-level design of the hardware substrate. We first present the overall circuit architecture and the implementation of the RRAM crossbar arrays, and then introduce a new dual-path configuration of the crossbar array to facilitate the realization of negative weights. Finally, a scaling-compatible implementation of the NAF based on the native VTC of CMOS logic circuits is presented.

A. NeuADC Circuit Architecture

The overall circuit architecture of the proposed NeuADC that realizes a three-layer MLP is illustrated in Fig. 4(a). We use RRAM crossbar arrays and inverter circuits at each layer to perform the basic NN operations (VMM and NAF) in the analog domain. The input analog signal represents the single "place holder" neuron in MLP's input layer. Hence, the weight matrix dimensions are $1 \times H$ between the input and the hidden layer and $H \times M$ between the hidden and the output layer, assuming there are H and M neurons in the hidden and output layers. In order to accommodate negative weights, we propose a new dual-path configuration such that each NN layer consists of a positive path and a negative path and each path requires

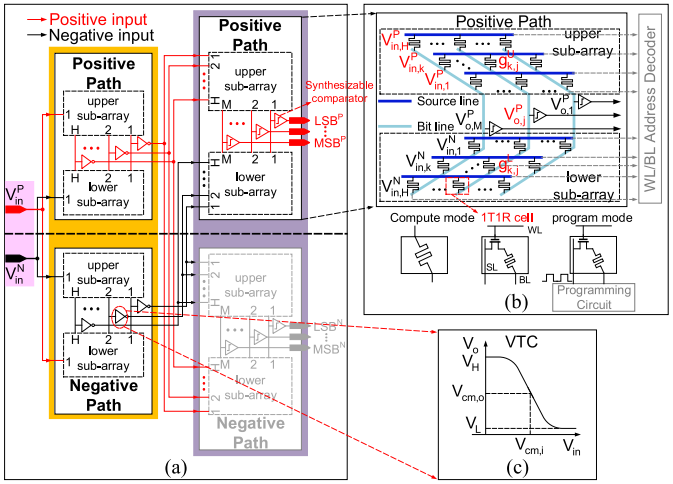


Fig. 4. NeuADC hardware implementation. (a) Dual-path architecture of NeuADC. (b) Zoomed-in RRAM $H \times M$ subarray. (c) Inverter VTC.

two RRAM subarrays, which are called the upper and the lower subarray. The operation of this dual-path configuration is explained later, but essentially the dual paths allow us to generate a pair of complementary voltage signals to represent the output of the VMM computation in the analog domain from the complementary input voltages.

B. RRAM Crossbar Array

A zoomed-in RRAM crossbar array working in compute mode is illustrated in Fig. 4(b). The $1 \times H$ complementary input vector represented by input voltages $V_{\text{in},1}$ to $V_{\text{in},H}$ are fed to each source line (SL) of the RRAM array, and each element in the weight matrix is stored as the conductance of the RRAM device in each weight cell. The computed VMM output vector appears as voltages at each column on the BLs. The weight cell consisting of one transistor and one RRAM device (1T1R) can operate in both compute mode and program mode. In the compute mode, the transistors in each 1T1R cells are turned on to select all the cells to compute in the analog domain by summing currents on BLs simultaneously. In the program mode, the RRAM device can be programmed to the desirable conductance by the programming circuits (PCs) and the address decoders (AD-DECs) shown in Fig. 4(b). In this mode, the access transistor plays a role in isolating individual cell for weight programming to prevent the sneak path. The area and power overheads of the access transistors are accounted for in Table III. We adopt the tuning mechanism of the program-and-verify (P&V) method to significantly reduce power consumption and improve the tuning resolution [32]. The proposed RRAM crossbar implementation differs from previous work introduced in Section II, as it obviates the use of analog-style circuits, such as Op-Amps and analog inverters, hence is much more scaling-compatible and synthesis-friendly.

C. Dual-Path Configuration

One critical difficulty of using RRAM crossbar as analog VMM is its inability to support negative weights, since the BL currents can only be added but not subtracted. Prior work proposes the use of analog inverters to circumvent this problem, but does not offer circuit implementation details [15]. We propose a new dual-path configuration that uses a pair of complementary voltage signals with opposite polarity and two signal paths, each consisting of two RRAM crossbar subarrays to overcome the negative weight challenge. To explain the dual-path operation and derive the voltage signal expression, we use the positive path crossbar shown in Fig. 4(b) as an example. First, assume there are H pairs of complementary inputs

$$V^P_{\text{in},k} = V_{\text{in},k}, \quad V^N_{\text{in},k} = V_{\text{DD}} - V_{\text{in},k} \tag{6}$$

where V_{DD} is the power supply, $k = 1, 2, \ldots, H$. We represent the output voltages at the crossbar BL on the positive path as $V^P_{o,j}$, and on the negative path as $V^N_{o,j}$. Hence, there are M pairs of outputs as $j = 1, 2, \ldots, M$. Each of $V^P_{\text{in},k}$ and $V^N_{\text{in},k}$ contributes to the output $V^P_{o,j}$ separately. Applying Thevenin theorem, the contribution of each pair of inputs superimposes to obtain the output BL voltages

$$V^P_{o,j} = \sum_{k=1}^{H} \left(W^{PP}_{kj} \cdot V^P_{\text{in},k} + W^{PN}_{kj} \cdot V^N_{\text{in},k} \right) \tag{7}$$

where $W^{PP}_{kj} = g^U_{kj} \cdot \epsilon$, $W^{PN}_{kj} = g^L_{kj} \cdot \epsilon$, and $\epsilon = 1/\sum_{l=1}^{H}(g^U_{lj} + g^L_{lj})$. The first superscript of W^{PP}_{kj} denotes which path the weight belongs to and the second superscript denotes which complementary input the weight acts upon. The superscript of g^U_{kj} denotes which subarray the conductance belongs to. By replacing $V^P_{\text{in},k}$ and $V^N_{\text{in},k}$ with (6), the output voltage $V^P_{o,j}$ of the positive path in (7) can be derived as $V^P_{o,j} = \sum_{k=1}^{H} W^P_{kj} \cdot V_{\text{in},k} + V^P_{\text{off},j}$. Here

$$W^P_{kj} = W^{PP}_{kj} - W^{PN}_{kj} = \left(g^U_{kj} - g^L_{kj} \right) \cdot \epsilon \tag{8}$$

and $V^P_{\text{off},j} = \epsilon \cdot \sum_{k=1}^{H} W^{PN}_{kj} \cdot V_{\text{DD}}$. It shows that thanks to the complementary voltage inputs that act on the subarrays, the effective weight W^P_{kj} is a subtraction of two conductances and thus can be negative. To make the proposed conductance subtraction scheme work properly, $V^N_{o,j}$, the complementary version of $V^P_{o,j}$, need to be generated, and it is achieved by incorporating the negative path where the polarity of input voltage pair is flipped. Thus, the expression of $V^N_{o,j}$ can be similarly derived as $V^N_{o,j} = \sum_{k=1}^{H} W^N_{kj} \cdot V_{\text{in},k} + V^N_{\text{off},j}$. Here

$$W^N_{kj} = W^{PN}_{kj} - W^{PP}_{kj} = \left(g^L_{kj} - g^U_{kj} \right) \cdot \epsilon \tag{9}$$

and $V^N_{\text{off},j} = \epsilon \cdot \sum_{k=1}^{H} W^{PP}_{kj} \cdot V_{\text{DD}}$. As Fig. 4(a) illustrates, our NeuADC implementation does not require the negative path at the output layer. Therefore, the negative path of the output layer is grayed out.

D. VTC-Based Activation Function

Past implementations of hardware NN circuits often use digital LUT or custom-designed analog neurons [15], [28] to approximate the ideal sigmoid function. However, modern deep learning techniques have shown that many different forms of NAF can result in successfully trained NN [29]. Therefore, we leverage the native VTC curve of CMOS logic circuits [46] to perform the NAF function in the NN hardware substrate, which effectively reduces the hardware complexity compared to traditional neurons [15], [28]. As depicted in Fig. 4(c), VTC exhibits saturation at both ends of the input range, and can be considered as a flipped-and-shifted version

of a general S-shaped curve, similar to the commonly used sigmoid function. To provide flexibility to the training process, current-starved inverters are used as the NAF in the hidden layer, because it allows the VTC curve to float in a range defined by V_H and V_L. Here, V_H and V_L are the highest and lowest voltage of VTC. The synthesizable comparators implemented with a three-input NAND gate [5], [6] are used in the output layer, since final digitization of output has to be performed. Both the inverter and the three-input NAND comparator-based neuron implementations are scaling-compatible and synthesis-friendly, thus significantly reduce the circuit complexity.

V. Training Framework

After designing the hardware substrate to express a general class of NN functions, we now introduce an offline training procedure that can automatically discover the circuit-level design parameters—the RRAM conductances, so that the circuit instantiates a function that well approximates the ideal quantization, to convert the input analog voltage to the correct output digital codes. To do this, we first define an NN model that corresponds to our hardware substrate, and identify the associated feasibility constraints on the NN weights to ensure they can be translated to a physically realizable circuit. We then introduce and discuss a new bit-encoding scheme that allows our hardware substrate to achieve finer quantization levels with alleviated circuit complexity. Next, we describe a method for optimizing these weights using a largely standard approach to gradient descent-based learning, but with modifications to enforce the feasibility constraints unique to our setting. We also incorporate nonidealities of devices into training to make NeuADC robust to PVT. Finally, we present how to instantiate RRAM conductance from the trained parameters.

A. Learning Objective

We model the input-output relationship of the NeuADC hardware substrate as a three-layer MLP with a single hidden layer

$$\begin{aligned} \tilde{h} &= L_1(V_{\text{in}}; \theta_1), \quad h = \sigma_{\text{VTC}}\left(\tilde{h}\right) \\ \tilde{b} &= L_2(h; \theta_2), \quad b = \tilde{b} > 0. \end{aligned} \tag{10}$$

Here, V_{in} is the scalar input signal and b is the final vector of output bits; $\tilde{h}$ denotes voltages at the output of the first crossbar layer, and modeled as a linear function L_1 of V_{in} with parameters θ_1 which corresponds to crossbar conductances. Each of these voltages is passed through an inverter, whose input-output relationship is modeled by the nonlinear function $\sigma_{\text{VTC}}(\cdot)$, to yield the vector h. The linear function L_2 models the second layer of the crossbar to produce another vector $\tilde{b}$, of size equal to the number of output bits. The final output bit-vector b is obtained by thresholding: yielding 0 for each element of $\tilde{b}$ that is below 0 and 1 otherwise.

The learning objective is to find optimal θ_1 and θ_2 such that for all values of V_{in} in the input range, NeuADC yields the corresponding bit-vectors b equal or close to the desired "ground-truth" vectors b_{GT}. We define a cost function that measures the discrepancy between predicted b and true b_{GT}. Note that the hard-thresholding to yield b from $\tilde{b}$ in (10) is non-differentiable, and this prevents propagating gradients to the parameters θ_1 and θ_2. Therefore, we use a differentiable cost $C(\tilde{b}, b_{\text{GT}})$ defined in terms of the unthresholded bit-vector $\tilde{b}$. Now, given a set $\{V_{\text{in}}^t, b_{\text{GT}}^t\}_t$ of pairs of signal and bit-vector values, the goal of training can be formally stated as solving the following optimization problem:

$$\left[\theta_{1,2}\right] = \arg\min \sum_t C\left(L_2\left(\sigma_{\text{VTC}}\left(L_1\left(V_{\text{in}}^t, \theta_1\right)\right), \theta_2\right), b_{\text{GT}}^t\right). \tag{11}$$

B. Circuit Model for Training

After sketching out the learning objective, we now provide details of the model in (10) to accurately reflect the hardware substrate described in Section IV. The first layer in our crossbar model has dual-path, each with H outputs. We denote these outputs as vectors $\tilde{p}$ and $\tilde{n}$, with $\tilde{h} = [\tilde{p}^T, \tilde{n}^T]^T$ being a $2H$ dimensional vector. Then, we define the linear relationship L_1 between these outputs as $\tilde{p} = W_1 \cdot V_{\text{in}} + V_1$, $\tilde{n} = V_{\text{DD}} - \tilde{p}$, which is equivalent to

$$\begin{aligned} \tilde{p} &= \max(0, W_1) \cdot V_{\text{in}} + \max(0, -W_1) \cdot (V_{\text{DD}} - V_{\text{in}}) \\ &\quad + (V_1 - \max(0, -W_1) \cdot V_{\text{DD}}) \\ \tilde{n} &= \max(0, -W_1) \cdot V_{\text{in}} + \max(0, W_1) \cdot (V_{\text{DD}} - V_{\text{in}}) \\ &\quad + (V_{\text{DD}} - V_1 - \max(0, W_1) \cdot V_{\text{DD}}) \end{aligned} \tag{12}$$

for the dual-path crossbar model in Section IV-C. Here, the learnable parameters are $\theta_1 = \{W_1, V_1\}$, where W_1 and V_1 are both H-dimensional vectors. Additionally, since these models will be instantiated using the RRAM crossbar array, it has the following practical feasibility constraints on W_1 and V_1:

$$\begin{aligned} 0 &\le V_1 - \max(0, -W_1) \cdot V_{\text{DD}} \\ &\le V_{\text{DD}} \cdot (1. - \text{abs}(W_1)) \\ 0 &\le V_{\text{DD}} - V_1 - \max(0, W_1) \cdot V_{\text{DD}} \\ &\le V_{\text{DD}} \cdot (1. - \text{abs}(W_1)). \end{aligned} \tag{13}$$

To obtain a high-fidelity VTC that matches well with the circuit-level behavior of the inverter, we perform a SPICE simulation at finely sampled input voltages.[2] We then formulate the function σ_{VTC} through linear interpolation between these sampled points, which ensures that we have both a value and a gradient for any input to the function. The final output function L_2 simply maps the inverter outputs $p = \sigma_{\text{VTC}}(\tilde{p})$ and $n = \sigma_{\text{VTC}}(\tilde{n})$ to the unthresholded bit vector $\tilde{b}$ as $\tilde{b} = \max(0, W_2)(p - V_{\text{cm},o}) + \max(0, -W_2)(n - V_{\text{cm},o}) + V_2$, with learnable parameters $\theta_2 = \{W_2, V_2\}$. Here, W_2 is an $M \times H$ matrix and V_2 is an M-dimensional vector, with M the number of output bits. Note that we define these parameters with respect to the hidden activations p and n after subtracting the mid-point voltage $V_{\text{cm},o}$ of the inverter output range, because this leads to more stable convergence during training.

C. Bits Encoding and Decoding Scheme

Our investigation suggests that the encoding scheme plays an important role in determining the convergence of the training process and ultimately the quantization quality of the NeuADC circuits. In this paper, bits encoding and decoding refer to the mapping of the analog input voltage levels to a specific digital code and converting it back to an analog value.

[2]Our approach to accounting for the effect of loading effects is described later in Section V-D.

1) Binary Encoding: Standard binary encoding is a straightforward way to define the ground-truth vectors b_{GT}

$$\sum_{i=1}^{M} 2^{i-1} \cdot b_{\text{GT}i} = \text{round}\left(\frac{V_{\text{in}} - V_{\text{min}}}{V_{\text{max}} - V_{\text{min}}} \times \left(2^M - 1\right) \right) \quad (14)$$

where V_{in} is the encoded form of input signal; V_{min} and V_{max} are the minimum and maximum values of the scalar encoded input signal V_{in}. A naive way to train the network is to interpret this as a classification task, and use a cross-entropy loss (i.e., interpret $\tilde{b}$ as logits and maximize log-likelihood) as

$$C\left(\tilde{b}, b_{\text{GT}}\right) = \sum_{i=1}^{M} b_{\text{GT}i} \log\left(1 + e^{-\tilde{b}_i}\right) + \left(1 - b_{\text{GT}i}\right) \log\left(1 + e^{\tilde{b}_i}\right). \quad (15)$$

However, this ignores the fact that we desire an accurate reconstruction of V_{in} as (14), and the accuracy of different weights have different effect on this accuracy. Based on this, we define two modified versions of the loss. The first simply weights each element of the standard loss by the contribution to the reconstruction of V_{in}

$$C_1\left(\tilde{b}, b_{\text{GT}}\right) = \sum_{i} 2^{i-1} \left[b_{\text{GT}i} \log\left(1 + e^{-\tilde{b}_i}\right) + \left(1 - b_{\text{GT}i}\right) \log\left(1 + e^{\tilde{b}_i}\right) \right]. \quad (16)$$

The second accounts for the fact that for some signal V_{in}, if a higher significant bit is incorrect and can not be corrected, flipping the values of the lower significant bits may lead to a better approximation [e.g., for ideal b_{GT} =1000 and the most significant bits (MSBs) of b stuck at 0, it is better to produce 0111 than 0000]. This effect is defined as $E_i(\tilde{b}, V_{\text{in}}) = |\hat{V}_{\text{in}} - b_{/i} - 2^{i-1}| - |\hat{V}_{\text{in}} - b_{/i} - 2^{i-1}|$, where

$$\hat{V}_{\text{in}} = \left(\frac{V_{\text{in}} - V_{\text{min}}}{V_{\text{max}} - V_{\text{min}}} \times 2^B \right) - 0.5$$
$$b_{/i} = \sum_{i' \neq i} 2^{i'-1} \left(\tilde{b}_i > 0\right). \quad (17)$$

Then, we define the second loss on $\tilde{b}$ as

$$C_2\left(\tilde{b}, V_{\text{in}}\right) = \sum_{i} \left[\max\left(0, -E_i\left(\tilde{b}, V_{\text{in}}\right)\right)^2 \log\left(1 + e^{-\tilde{b}_i}\right) + \max\left(0, E_i\left(\tilde{b}, V_{\text{in}}\right)\right)^2 \log\left)1 + e^{-\tilde{b}_i}\right) \right]. \quad (18)$$

Note that this second loss is defined in terms of the true signal V_{in} and not its binary encoding b_{GT}, although it implicitly assumes that b represents a binary encoding of V_{in}. In this setting, we train the network to minimize a weighted sum of the two losses ($\alpha \cdot C_1 + C_2$), with more weight placed on the second loss ($\alpha < 1$), while the first loss mainly serves to guide training in initial iterations. However, we find that although the modified cost function modestly improves the training, it remains quite hard to find a good approximation to this mapping (from V_{in} to its binary encoding vector) using a circuit with a limited number of hidden units. This is because, as depicted in Fig. 5(a), the binary encoding corresponds to a high-frequency target function for the least significant bits (LSBs), which must change signs 2^M times in the input range. Moreover, small errors in any of the more significant bits can cause large deviations in the reconstructed analog value.

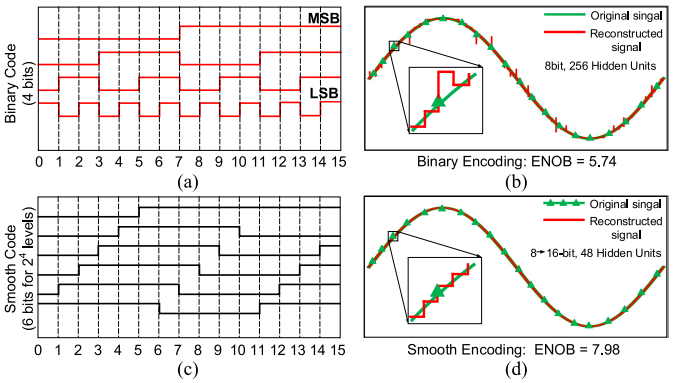


Fig. 5. (a) Transition of different bits in a binary code as its digital value changes. (b) Example of reconstructed waveform from NeuADC outputs trained with binary bits encoding. (c) Transition of different bits in our proposed smooth code. (d) Example of reconstructed waveform from NeuADC outputs trained with smooth-bit encoding.

2) Smooth Encoding: Accordingly, we propose the use of "smooth" $A \rightarrow B$ codes that replace an A-bit binary encoding with $B > A$ bits. These codes represent each of the 2^A levels with B-bit unique codewords that have the following two properties: 1) only one bit changes between subsequent levels (this property is similar to those of "Gray codes") and 2) each bit flips a minimum number of times. Given parameters A and B, these codewords are automatically constructed by beginning with an all-zero codeword for the lowest level, and then for each subsequent level, choosing to flip the bit that was least recently flipped. This leads to smoother bit functions with fewer transitions as shown in Fig. 5(c). Fig. 5(b) and (d) shows some example of reconstructed waveforms of an input sinusoidal signal by the circuits learned with binary and smooth codes, respectively. We find that the binary encoding is able to achieve reasonable but mediocre reconstructions with a large number of hidden units (∼256), whereas given a wide-enough bit-vector, smooth encoding can accurately reconstruct with far fewer hidden units (∼48). We train this encoding with the simple cross-entropy loss defined in (15), although we use a squared version (after summing over bits) to emphasize the penalty for multiple errors in the same sample.

3) Decoding Scheme: Given a trained circuit that produces bit-vectors b for different inputs V_{in}, we use a simple decoding scheme that constructs an LUT to find the corresponding analog value of each possible bit-vector. We use an 8-bit ADC to explain the concept and the proposed decoding scheme, as illustrated in Fig. 6. First, we use the final learned parameters to compute the sets $\{V_{\text{in}}, b\}$ for a finely sampled set of values V_{in}. In terms of an 8-bit ADC, there are 256 distinct bit-vectors $b_{[i]}, i = 0, 1, 2, \ldots, 255$ in b, and each $b_{[i]}$ corresponds to N_i finely sampled analog input $V_{\text{in}}(t_j), j = 1, 2, \ldots, N_i$. Therefore, the corresponding analog value of $b^t_{[i]}$ can be calculated as $V'_{b,i} = \sum_{j=1}^{N_i} V_{\text{in}}(t_j)/N_i$. We then repeat the same procedure to all the $b^t_{[i]}$ to construct the LUT $\{V'_b, b\}$ for each $b_{[i]}$ in b. We apply this for circuits trained with both encoding schemes, as it helps calibrate for deviations in the learned mapping.

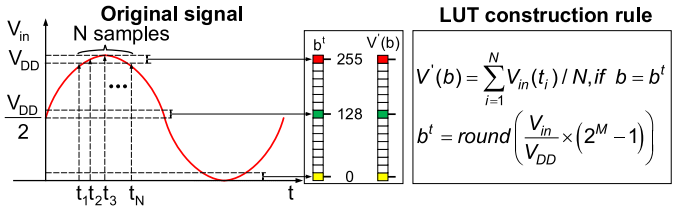


Fig. 6. Illustration of decoding scheme and the rule of constructing an LUT.

D. Hardware-Oriented Training

Having defined the mapping from V_{in} to the circuit output $\tilde{b}$, and a differentiable cost $C(\tilde{b}, b_{\text{GT}})$ on these outputs, we are now able to train the parameters using stochastic gradient descent [34]. We initialize the parameters θ_1 and θ_2 randomly, and update them iteratively based on gradients computed on mini-batches of $\{(V_{\text{in}}, b_{\text{GT}})\}$ pairs randomly sampled from the input range. However, this standard approach to NN training does not account for the feasibility constraints on θ_1 in (13). To enforce these constraints, we apply the following steps. In each iteration of training, we clip the positive and negative path biases [the third term in (12)] to match (13) such that the final layer does not learn to depend on an infeasible combination of inputs. Further, we also periodically (every 256 iterations) clip the parameters themselves to the feasible set, by clipping W_1 between $[-0.5, 0.5]$ and scaling V_1 accordingly.

The other issue we need to deal with is that the input-output relationship of the inverters depends on its loading impedance, which is determined by the conductances of the second layer. In other words, the exact curve of the σ_{VTC} function differs as the values of θ_2 vary, and this can only be determined through SPICE simulations. To deal with it, we first obtain σ_{VTC} assuming no loading impedance, and then train the circuit with a fixed σ_{VTC}. We then update σ_{VTC} by performing a second SPICE simulation, this time based on the current values of the learned parameters θ_2. We then continue training the network for a few more iterations, using the updated σ_{VTC}. We find that two rounds of this iteration are sufficient to yield circuit parameters that perform well with actual loading effect.

E. Variation-Aware Retraining

So far, we build an ideal training framework without accounting for the nonidealities of devices such as the variation effects. PVT variations can degrade the performance of CMOS devices. In this part, we focus on how to improve the robustness of NeuADC by incorporating the PVT variations of CMOS devices and the limited resolution of RRAM resistance into training. It allows us to obtain a group of robust trained parameters to design a hardware substrate that is immune to variations.

1) CMOS PVT Variations: In reality, neurons on the same hardware substrate experience different PVT variations, which makes the practical VMM computation on the hardware substrate mismatched with the simulation results of offline NN model and eventually leads to wrongly flipped digital bits in the output layer. To overcome such effects, a naive way is to incorporate all possible VTCs into training, covering all the process corners as illustrated in Fig. 7(a). Namely, we let each neuron in (10) randomly pick up a VTC from VTC group A_{VTC} (obtained by global Monte Carlo simulations) during each training epoch

$$\sigma_{\text{VTC}}^{i} = A_{\text{VTC}}[\text{rand int}(N)], i = 1, 2, \ldots, H. \tag{19}$$

However, we find it hard to achieve a good training performance with a reasonable size of NN. As Fig. 7(b) shows, the variation of VTC's switching threshold is more than 70 mV under a moderate global Monte Carlo simulation. Overcoming such variations requires a huge number of hidden neurons.

It turns out that in practice, it is relatively easy to figure out which process corner each chip belongs to through post-fabrication binning [40], [41]. Based on this fact, we propose a more efficient way to find several local optima to overcome the PVT variations with much less hardware resources and shorter training time. For example, given that the chip is in TT (typical nMOS and typical pMOS) corner, we first use local Monte Carlo simulation to find all possible VTCs and then incorporate them into training using the procedure in (19). Since the switching threshold variation of VTCs at TT corner is only 4.5 mV as shown in Fig. 7(d), it is easier to find a group of local optimal design parameters with much less size and shorter training time. We repeat the same procedure for the local region under each process corners in Fig. 7(c), and it is observed that only five LUTs are needed to cover the regions across all process corners.

2) RRAM Variations: The limited RRAM resistance resolution corresponds to the limited resolution of weights. We solve this issue by constraining the resolution of weight in N-bit together with incorporating VTC variations during retraining. Regarding the issue of RRAM process variations [31], we look at the median ENOB of different retrained NeuADC circuits after perturbing their conductances with log-normal noise of different standard deviations σ—i.e., multiplying them by $\exp(\epsilon)$ where $\epsilon \sim \mathcal{N}(0, \sigma^2)$. The corresponding performance

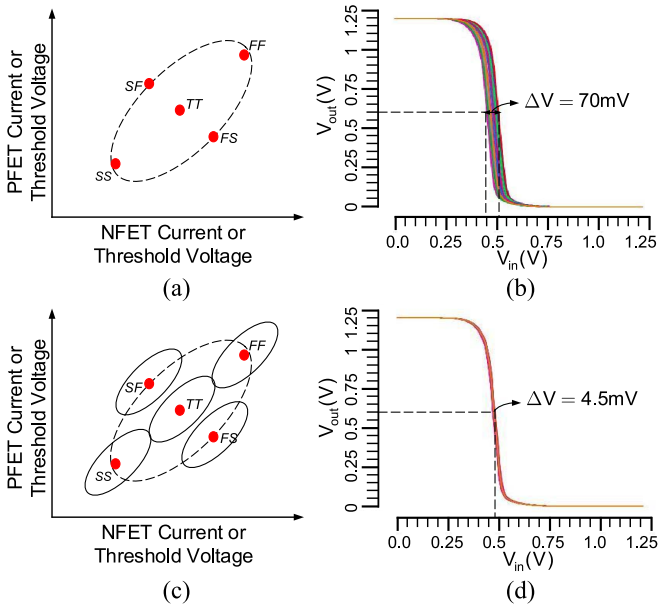


Fig. 7. (a) Conceptual illustration of the feasible region to cover all five process corners variation in corner plane. (b) Variation of VTC at $V_{\text{DD}} = [1.17\text{ V}, 1.23\text{ V}]$, $T = [-40\,^{\circ}\text{C}, 80\,^{\circ}\text{C}]$ under all five process corners by using 100 times Monte Carlo simulation. (c) Proposed method to cover most region in the corner plane. (d) Variation of VTC at $V_{\text{DD}} = [1.17\text{ V}, 1.23\text{ V}]$, $T = [-40\,^{\circ}\text{C}, 80\,^{\circ}\text{C}]$ under TT corner by using 100 times Monte Carlo simulation. The simulation of (b) and (d) is based on 130-nm CMOS technology.

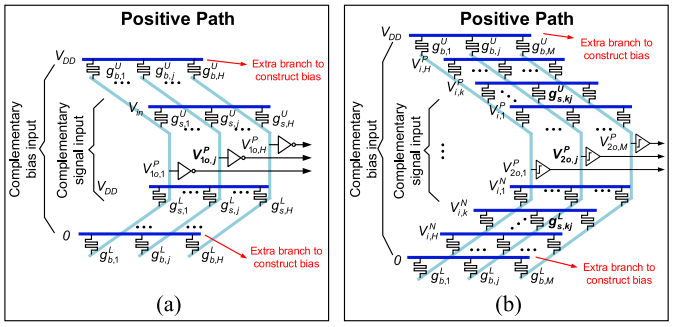


Fig. 8. Hardware model to map with the training model. Crossbar array of positive path for the (a) first layer and (b) second layer. Note that the 1R cell is the simplified representation of 1T1R cell.

degradation for different NeuADC designs is evaluated later in Section VII-C.

F. Conductance Instantiation

After learning the optimal parameters θ_1 and θ_2 through the hardware-oriented training process, we translate them into RRAM conductance. In addition to the weights, precisely-trained biases need to be realized. We tackle this by adding an extra row in the RRAM crossbar in both the hidden layer and the output layer as illustrated in Fig. 8. We instantiate the biases in (12) by providing the supply voltage $V_{\rm DD}$ as an input, in addition to the signal inputs $V_{\rm in}$ and $V_{\rm DD} - V_{\rm in}$. As an example, we begin by considering the first layer. For each output of the positive path, i.e., $\tilde{p}_i,\ i \in \{1, \ldots, H\}$, we denote $g^U_{s,i}, g^L_{s,i}, g^U_{b,i}$, and $g^L_{b,i}$ as the conductances connecting the output to $V_{\rm in}$, $V_{\rm DD} - V_{\rm in}$, the supply $V_{\rm DD}$ and ground as illustrated in Fig. 8(a). After accounting for this modification to the original crossbar, (7) can be rewritten as

$$V^P_{1o,j} = W^{PP}_j \cdot V_{\rm in} + W^{PN}_j \cdot (V_{\rm DD} - V_{\rm in}) + W^b_j \cdot V_{\rm DD}. \quad (20)$$

Here, $W^{PP}_j = g^U_{s,j}/C_1, W^{PN}_j = g^L_{s,j}/C_1, W^b_j = g^U_{b,j}/C_1$, and $C_1 = g^U_{s,j} + g^L_{s,j} + g^U_{b,j} + g^L_{b,j}$. Combine (12) and (20), these conductances can be related to the learned weights W_1 and V_1 as

$$\begin{aligned} g^U_{s,j} &= C_1 \times \max\big(0, W_{1,j}\big) \\ g^L_{s,j} &= C_1 \times \max\big(0, -W_{1,j}\big) \\ g^U_{b,j} &= C_1 \times (V_1 - \max(0, -W_1) \times V_{\rm DD})/V_{\rm DD} \\ g^L_{b,j} &= C_1 - g^U_{s,j} - g^L_{s,j} - g^U_{b,j}. \end{aligned} \quad (21)$$

Since C_1 is a scaling factor, it should be chosen to ensure the RRAM conductances fall into a reasonable range. The same process can be repeated to instantiate the conductances for the negative path in the first layer. For the second layer, we adopt a similar strategy, but first normalize both W_2 (and proportionally V_2) such that the sum of the positive and negative values across all columns is less than magnitude 1

$$W'_2 = \frac{W_2}{\beta \cdot \sum(\text{abs}(W_2), 0)}, V'_2 = \frac{V_2}{\beta \cdot \sum(\text{abs}(W_2), 0)} \quad (22)$$

where $\sum(\text{abs}(W_2), 0)$ means the summation of all the elements (their absolute value) in the same column; β is a scaling factor large than 1. The second layer conductances can then be computed as

$$\begin{aligned} g^U_{s,kj} &= C_2 \times \max\Big(0, W'_{2,kj}\Big) \\ g^L_{s,kj} &= C_2 \times \max\Big(0, -W'_{2,kj}\Big) \\ g^U_{b,j} &= \frac{C_2 \times \Big(-\sum_k \Big(W'_{2,kj} \times V_{{\rm cm},o}\Big) + V'_{2,j} + V_{{\rm cm},i}\Big)}{V_{\rm DD}} \\ g^L_{b,j} &= C_2 - g^U_{s,kj} - g^L_{s,kj} - g^U_{b,j} \end{aligned} \quad (23)$$

where $V_{\rm cm,i}$ and $V_{{\rm cm},o}$ are input and output mid-point voltage of the inverter input and output range, respectively. C_2 is also a scaling factor chosen to ensure the RRAM conductances fall into a reasonable range.

TABLE I
SIMULATION CONFIGURATION PARAMETERS

Training Parameters	**Description**
Optimizer	Adam
Batch size	4096
Projection step	256
# of iterations	$5.12\times\ 10^4$
Learning rate	$[10^{-3}, 10^{-4}]$
Training constant α (binary encoding)	0.125,0.25,0.5
Encoding coefficient (logarithmic)	$a = b = c = 1, d = 0$
Encoding coefficient (square-root)	$a = c = 1, b = d = 0$
Technology Parameters	**Description**
CMOS technology	CMOS $130nm$
Process variation	$ss/tt/ff/sf/fs$
Voltage variation	$1.17V \sim 1.23V$
Temperature variation	$-40°C \sim 80°$C
RRAM technology	HfOx-based RRAM
RRAM tunneling gap	$0.2nm \sim 1.9nm$
RRAM resistance resolution	5-bit$\sim$10-bit, step = 1-bit
RRAM resistance range	$290\Omega \sim 500$kΩ
RRAM resistance variation (σ)	$[0, 0.1]$, step =0.025

G. Quantization Schemes

We have successfully formulated the NeuADC design as a learning problem. The same training framework can be extended beyond the normal linear uniform quantization scheme to learn parameters for other quantization schemes tailored to the desired precision requirements for specific applications. To accommodate alternative schemes, the only update needed is changing the definition of $b_{\rm GT}$ in (14) by using a function of $V_{\rm in}$ instead of $V_{\rm in}$ itself. For logarithmic encoding, $V_{\rm in}$ is defined as

$$V_{\rm in,log} = c \cdot \log_2(a \cdot V_{\rm in} + b) + d \quad (24)$$

whereas for the square root encoding, $V_{\rm in}$ is defined as

$$V_{\rm in,sq} = c \cdot \sqrt{a \cdot V_{\rm in} + b} + d. \quad (25)$$

Here, a, b, c, and d are the quantization encoding coefficients. The detailed value of these coefficients for different encoding schemes are listed in Table I.

VI. SIMULATION METHODOLOGY

In this section, we present the detailed methodology used in our simulation setup to design, train/synthesize, and evaluate the proposed NeuADC circuits. We first summarize the automated design flow enabled by our novel NeuADC design

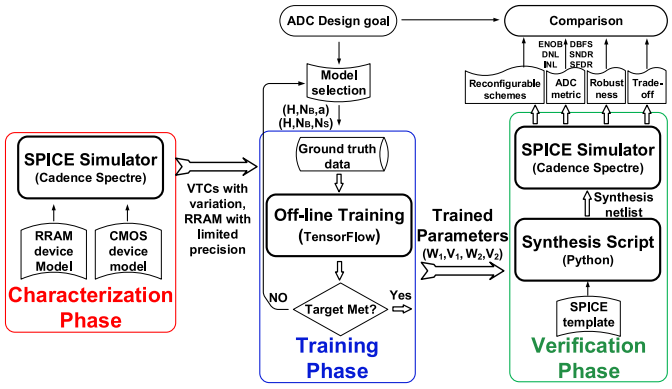


Fig. 9. Design automation flow.

approach. We then present the configurations used in our training setup. Finally, we show the technology model for the simulation infrastructure.

A. NeuADC Design Flow

The fully automated design flow based on the NeuADC design approach is presented in Fig. 9, which consists of three phases—characterization, training, and verification.

1) Characterization Phase: Once the RRAM device model and CMOS device model are prepared, basic characterization at device and circuit level is first performed on commercial SPICE simulator (e.g., Cadence Spectre). By SPICE simulation, the characterization data, such as RRAM conductance/resistance precision and CMOS inverter VTCs with variation, are extracted. The RRAM resistance resolution is set between 5 and 10 bits, with 1-bit step. VTCs are obtained by running 1000 times Monto Carlo simulation with supply voltage $V_{\mathrm{DD}} = [1.17\ \mathrm{V}, 1.23\ \mathrm{V}]$ and temperature $T = [-40\ ^{\circ}\mathrm{C}, 80\ ^{\circ}\mathrm{C}]$ for each process corner. Then, these characterization data are fed to the training framework. During the training, VTCs are utilized as the NAF, while the RRAM conductance/resistance precision is applied to constraint the precision of weights.

2) Training Phase: Given the desired ADC goals, the NN model of the NeuADC circuit can be fully captured by a group of hyper-parameters based on the proposed NN-inspired realization of A/D converters. For binary-encoding, the hyper-parameters are (H, N_B, and α). For smooth-encoding, the hyper-parameters are (H, N_B, and N_S). Here, H denotes the number of hidden neurons; N_B and N_S denotes the number of binary bits and the number of smooth bits, and α is training constant for binary-encoding. The ground-truth datasets can be generated according to the desirable A/D quantization with a resolution of $1/2^{N_B}$. The offline training framework then uses the datasets together with the device variation statistics to train the MLP network that models the behavior of the NeuADC circuit for each quantization scheme. During the training, the VTC of each neuron is randomly picked in each epoch. The training iterations are monitored to ensure the convergence of the learning. The reconstruction quality is verified at the end of each training. If the reconstructed signals match well with the labeled ground-truth signals, the trained model parameters (W_1, V_1, W_2, V_2) are saved for later verification using SPICE simulation. Otherwise, updated hyper-parameters will be used to train a new model until satisfactory performance is met.

3) Verification Phase: After training, the trained model parameters (W_1, V_1, W_2, and V_2) are fed into the synthesis script. Then the synthesis script instantiates the device/circuit design parameters, such as RRAM conductance and inverter sizes based on the device SPICE template, so that the SPICE netlist of NeuADC is automatically synthesized. The synthesized netlist allows us to perform a comprehensive sets of circuit analysis using SPICE simulator to rigorously evaluate and verify the performance of the NeuADC circuits. In our simulations, we first evaluate the reconfigurable quantization schemes of NeuADC. Then, the typical ADC metrics, such as effective number of bits (ENOB), differential nonlinearity (DNL), integral nonlinearity (INL), frequency spectrum analysis, and signal to noise and distortion ratio (SNDR) are evaluated. Third, we assess the robustness of NeuADC against device nonidealities, such as PVT variations and limited RRAM precision. Finally, these simulation metrics are compared with initial design goals to help us optimize design parameters.

B. Training and Simulation Configurations

1) Training Setup: We use TensorFlow for the NN training. The NeuADC NN model is trained via stochastic gradient descent with the Adam optimizer [34]. We choose a batch-size of 4096 samples, and apply the projection step performed on the weights W_1 and V_1 every 256 iterations. We train for a total of 5.12×10^4 iterations (except for certain smooth code NeuADC models that converge much faster), varying the learning rate from 10^{-3} to 10^{-4} across iterations. For the binary encoding models, we train three versions of each NeuADC circuit with $\alpha = 0.125$, 0.25, and 0.5, and choose the one that yields the best results. Encoding coefficients for logarithmic encoding are set as $a = b = c = 1$ and $d = 0$, while for square-root encoding, the coefficients are set as $a = c = 1$ and $b = d = 0$.

2) Technology Model: RRAM and CMOS transistors are the two device elements used in our NeuADC hardware substrate. We use HfOx-based RRAM device model [15], [20] to implement the crossbar array. The transistor model is based on a 130-nm CMOS technology. The inverters, the output comparators, and the transistor switches in the RRAM crossbars are all simulated with the 130-nm transistor model in Cadence Spectre. CMOS and RRAM device resolution and variation are included into training. We choose a moderate variation range $[0, 0.1]$ in our evaluation from a broad range of RRAM literature [42]–[44]. Configuration parameters from both the training setup and the technology model are summarized in Table I. All the simulation results are presented in Section VII.

VII. Simulation Results

We perform a comprehensive set of simulations using the simulation methodology explained in Section VI to thoroughly evaluate the performance of our proposed NeuADC design approach. The results discussed in this section are based on SPICE-level circuit simulations using 130-nm CMOS technology. Note that the simulations in Sections VII-A and VII-B are performed considering only PVT condition (TT, 1.2 V, 27 °C) with fixed RRAM resolution (9 bits) and minor RRAM stochastic variation, whereas the robustness evaluation in Section VII-C look at the performance spread under all PVT conditions at fixed TT process corner.

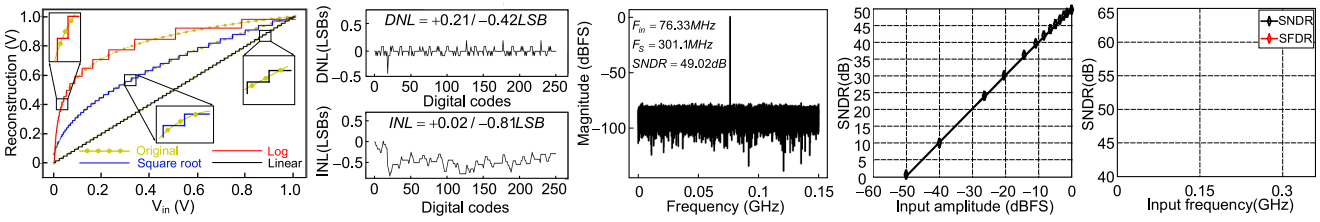


Fig. 10. (a) 6→8 bits, 12 hidden units NeuADC multiquantization example. (b)–(d) Simulated metrics of 8→16 bits, 48 hidden units NeuADC. (b) DNL and INL. (c) Output spectrum with a −0.5 dBFS, 76.33-MHz input, and 303.1-MHz sampling rate. (d) Linear SNDR trend with increasing input amplitudes. (e) SNDR and SFDR trend with increasing input frequency.

TABLE II
LEARNED PERFORMANCE OF NEUADC WITH BINARY AND SMOOTH ENCODINGS MEASURED BY ENOB

With Binary Codes			With Smooth Codes		
# Bits	# Hidden	ENOB	# Bits	# Hidden	ENOB
4	16	3.46	6 → 8	12	5.95
4	64	3.73	6 → 8	16	5.98
4	128	3.74	6 → 12	12	5.99
6	24	4.01	7 → 12	12	5.21
6	96	4.07	7 → 15	12	6.91
6	192	4.48	7 → 15	16	7.00
8	32	4.76	8 → 16	32	4.64
8	128	5.51	8 → 16	48	7.98
8	256	5.74	8 → 16	64	7.99

TABLE III
PERFORMANCE COMPARISON

	ISSCC14' [38][c]	Sto1 [5][c]	Sto2 [6][c]	This work[d]
Supply Voltage (V)	1.1	1.3	1.2	**1.2**
Technology (nm)	40	180	90	**130**
Area (mm^2)	0.052	0.43	0.18	**0.01/0.2**
Power (mW)	27.4	0.631	34.8	**18/30**
f_S (S/s)	2.2G	8M	0.21G	**1G/0.3G**
BW (Hz)	1.1G	4M	0.105G	**0.5G/0.15G**
Resolution (bits)	7	6	6	**6/8**
ENOB (bits)	5.94	5.28	5.08	**5.95/7.96**
FoM_W (fJ/c)[a]	205.7	1970	3255	**291/401**
FoM_S (dB)[b]	143.4	132	121	**141.8/149**
Auto. Synthesis?	No	Yes	Yes	**Yes**
Reconfigurable?	No	No	No	**Yes**

[a] $FoM_W = P/(2^{ENOB} \cdot f_s)$. The smaller the value, the better the performance. P is power.
[b] FoM_S = SNDR + $10 \cdot \log_2(BW/P)$). The larger the value, the better the performance.
[c] Measured results based on fabricated chip.
[d] Simulation results based on SPICE.

A. Signal Reconstruction Validation

We first exhibit the reconstruction ability of NeuADC under three different quantization schemes and illustrate the reconstructed signals in Fig. 10(a). We pick a specific NeuADC model (6→8 bits, 12 hidden units) and train it with the three groups of ground-truth data in Fig. 3(c). For each quantization scheme, we then reconstruct the signal using the decoding scheme in Section V-C. The reconstructed signals (labeled as linear, square root, and log, respectively) match well with the original signal (labeled as original) shown in yellow under different schemes, demonstrating that NeuADC can perform high-fidelity signal reconstruction with multiple reconfigurable quantization support using exactly the same hardware substrate.

Then, we demonstrate the reconstruction of NeuADC based on the reconstruction V_b' of a sinusoidal input signal V_{in} in linear uniform encoding at 100-KHz frequency. We report ENOB of the reconstructed waveform using its standard definition ENOB = (SNDR − 1.76)/6.02, where SNDR is obtained from analyzing NeuADC's output spectrum with fast Fourier transform (FFT). Table II lists the quantization quality measured by ENOB of several NeuADC designs with different number of output bits and hidden neurons using both binary and smooth codes.

The advantage of smooth encoding is clearly shown in Table II. Binary code requires a large number of hidden neurons and its ENOB plateaus around 5.74 even when the hidden neuron size is increased to 256 for 8-bit digital outputs. Despite its coding redundancy, smooth code can achieve much better ENOB with much fewer number of hidden neurons. For example, the 7→12 smooth code model with 12 hidden neurons exhibits an ENOB of 5.21. However, as the output neurons increase to 15, the ENOB increases rapidly to 6.91, and further increasing the hidden neurons to 16 could recover the ENOB fully to its theoretical upper bound of 7 for a 7-bit ADC. It suggests there exists an interesting relationship between the encoding redundancy, the number of hidden neurons, and the total size of the NN models employed by NeuADC, which plays a critical role in alleviating the exponential increase of hardware complexity as observed in conventional flash ADCs.

B. ADC Metric Evaluation

Classic ADC designs are evaluated for a number of different metrics. In this evaluation step, we measure our NeuADC circuits against many established ADC metrics to demonstrate its practical value. We choose a specific NeuADC model (8→16 bits, 48 hidden units) as an example for evaluation. DNL and INL are typically used to characterize ADC's static performance. Fig. 10(b) shows the DNL and INL of our proposed NeuADC based on SPICE simulation. The worst DNL and INL are −0.42LSB and −0.81LSB, respectively, well within the range of conventional linearity requirements. The simulated output spectrum is then used to characterize the dynamic performance of the ADC. Fig. 10(c) shows a −0.5 dB to full scale (dBFS) of a 76.33-MHz input. The SNDR is 49.02 dB. The trend of SNDR with changing input amplitude is plotted in Fig. 10(d) and follows a linear relationship. To evaluate the input bandwidth (BW) of NeuADC, we apply sinusoidal input signals with different frequencies to the NeuADC circuit. The trend is shown in Fig. 10(e). At 150 MHz, the SNDR experiences slight degradation of 0.22 dB, but remains acceptable at 49.7 dB, which translates to an ENOB of 7.96 bits. We therefore report this number as the BW of our design in Table III, which corresponds to

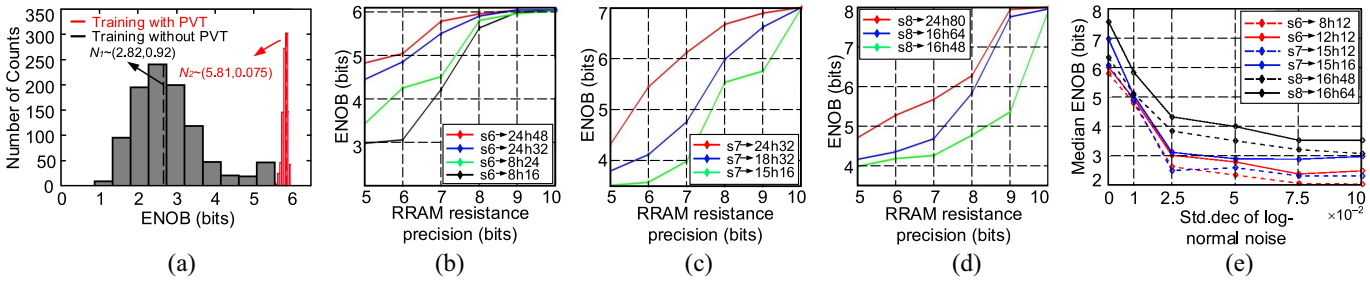


Fig. 11. (a) ENOB distribution comparison between with and without incorporating PVT variation into training by taking 6→8 bits, 12 hidden units NeuADC model at TT corner as an example. (b)–(d) Performance degradation for different NeuADC designs with decreasing RRAM resistance precision. (b) 6-bit models. (c) 7-bit models. (d) 8-bit models. (e) Performance degradation for different NeuADC designs with increasing log-normal noise at fixed RRAM resistance resolution of 9 bits.

300-MHz sampling frequency according to Nyquist–Shannon sampling theorem.

C. Robustness Against Devices Nonidealities

We first examine the performance of NeuADC with the ENOB metric under CMOS PVT variations. We compare the ENOB with and without incorporating PVT variations into training by selecting the 6→8 bits, 12 hidden units NeuADC model at TT corner for illustration. The comparison is illustrated in Fig. 11(a). Without incorporating PVT variations into training, the distribution of ENOB under 1000 Monte Carlo simulation runs centers around 2.82 with a large standard deviation of 0.92, resulting in poor quantization performance. After incorporating PVT variations of VTC into training, the distribution of ENOB under 1000 Monte Carlo runs is much more narrowly centered around 5.81 with merely 0.075 standard variation. The striking contrast shows that our variation-aware retraining can greatly improve the robustness of NeuADC.

We then evaluate the performance of the NeuADC given realistic considerations of RRAM device nonidealities. Previous work has employed 6–12-bit RRAM precision for RRAM-based NN computation [14], [15], [38], [45]. Since RRAM is still an emerging device with many active research and development efforts, we choose the RRAM resistance precision to be 5–10-bit in our evaluation. The ENOB versus RRAM precision curves are presented in Fig. 11(b) and (d). Each point shown in the figure is obtained from 1000 Monte Carlo simulation runs. We observe that to achieve a target resolution of A-bit ($6 \leq A \leq 8$) in NeuADC, an $(A+1)$-bit RRAM resistance precision is usually sufficient. Lower RRAM precision can degrade the resolution of the quantization and result in the use of unfavorably large NN size to compensate for the precision loss.

Finally, to examine the stochastic variation of RRAM, we look at the median ENOB of different NeuADC models with fixed RRAM precision (9 bits) after perturbing their resistance with log-normal noise in different standard deviations [30]. The results are presented in Fig. 11(e). For these experiments, we first incorporate both CMOS PVT variations and RRAM limited resistance resolution into training and then instantiate several batches of 100-run Monte Carlo simulations with different level of resistance variations as modeled by the standard deviation of the log-normal distribution (σ) and compute the median ENOB of each. We find that in general the learned

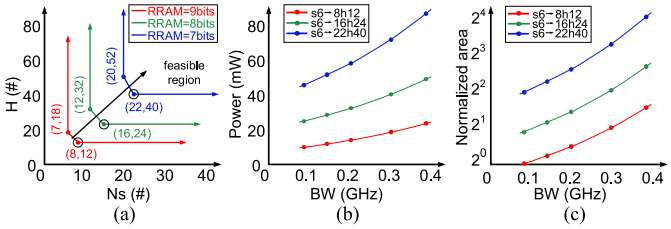


Fig. 12. Design tradeoff of NeuADC by taking 6-bit NeuADC model for example. (a) Hidden layer neuron number (H) versus output bits (N_S). (b) Power versus BW. (c) Normalized area versus BW.

NeuADC continues to perform reasonably well at moderate levels of noise, demonstrating robustness against the nonidealities in both CMOS and RRAM devices. Moreover, among two NeuADC designs that have similar performance with process variation, the one that uses more output bits or hidden units tends to exhibit more robustness against variation.

D. NeuADC Design Tradeoffs

We explore the design tradeoffs of NeuADC by taking 6-bit A/D as an example. As Fig. 12(a) shows, given a specific resolution of RRAM, there exists a lower bound combination of hyper-parameters (H, N_S) that converges to an ideal ENOB during training. However, the lower bound shifts toward up right as the resolution of RRAM decreases. This is because increasing the size of NN can improve the robustness of NeuADC performance. There also exist tradeoffs between BW and power consumption, as well as BW and area, as displayed in Fig. 12(b) and (c). As the BW increases, the power consumption of each model rises. The reason is that a wider BW means decreasing the RRAM resistance in the crossbar array, which results in increased power consumption. Also, in order to increase the BW, we need to strengthen the driving ability of neurons by sizing up the inverters. Therefore, the area of the NeuADC circuit increases with the BW. The area shown in Fig. 12(c) is normalized to a 6→8 bits, 12 hidden units NeuADC working at 0.1-GHz input signal frequency.

E. Comparison and Discussion

Finally, we compare the 6-bit (6→8 bits, 12 hidden units) and 8-bit (8→16 bits, 48 hidden units) NeuADC design with both the state-of-the-art manually designed ADC (ISSCC14' [37]) and other novel ADC architectures (Sto1 [5] and Sto2 [6]). ISSCC14' is a typical flash ADC that works

at a high sampling rate and achieves the state-of-the-art figure-of-merit (FoM). However, it is neither synthesiable nor configurable. Sto1 [5] and Sto2 [6] are the two representative synthesizable flash ADCs, but they still lack the reconfigurability to realize different quantization schemes. What is more, neither yields competitive advantages in FoM. The proposed 6-bit NeuADC can work at 1-GS/s sampling rate and achieve 291-fJ/c FoM$_W$ and 141.8-dB FoM$_S$. The proposed 8-bit NeuADC can work at 0.3-GS/s sampling rate and achieve 401-fJ/c FoM$_W$ and 149-dB FoM$_S$. The detailed comparisons are summarized in Table III. It demonstrates that NeuADC can deliver significantly superior performance (higher sampling frequency f_S, smaller FoM$_W$, and higher FoM$_S$) compared with alternative novel synthesizable ADC architectures (Sto1 [5] and Sto2 [6]). And its automated design flow can also achieve reasonable *FoMs* comparable to those of the state-of-the-art manually designed ADC (ISSCC14' [37]). Moreover, NeuADC can achieve reconfigurable quantization schemes. Please note that ISSCC14' [37], Sto1 [5], and Sto2 [6] are based on the measured results, while our results are obtained through simulation by using 130-nm CMOS technology. Our initial results suggest that further power saving and area reduction are feasible by scaling to smaller technology nodes.

VIII. Conclusion

We present NeuADC—a novel automated design approach for synthesizable A/D conversion with reconfigurable quantization support using the same hardware substrate. Inspired by NN, NeuADC is built upon a general NN hardware substrate enabled by a novel dual-path mixed-signal RRAM crossbar architecture and CMOS inverter as neuron. The design parameters are "learned" through NN training. We exploit a new smooth-bit encoding scheme to improve the training accuracy and develop hardware-oriented circuit models and constraint formulations in the training process. We also incorporate the nonidealities of devices into training to improve the robustness of NeuADC. The entire synthesis process of NeuADC can be automated without human designers in the loop. Comprehensive ADC performance metrics are evaluated using circuit-level SPICE simulation. Results demonstrate that our automatically synthesized NeuADC can indeed be reconfigured for different quantization schemes with high-fidelity reconstruction accuracy and achieve performance comparable to state-of-the-art ADCs despite limited RRAM resistance precision.

References

[1] R. H. Walden, "Analog-to-digital converter survey and analysis," *IEEE J. Sel. Areas Commun.*, vol. 17, no. 4, pp. 539–550, Apr. 1999.

[2] A.-J. Annema, B. Nauta, R. van Langevelde, and H. Tuinhout, "Analog circuits in ultra-deep-submicron CMOS," *IEEE J. Solid-State Circuits*, vol. 40, no. 1, pp. 132–143, Jan. 2005.

[3] D. Leenaerts, G. Gielen, and R. A. Rutenbar, "CAD solutions and outstanding challenges for mixed-signal and RF IC design," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, 2001, pp. 270–277.

[4] F. Gong, S. Basir-Kazeruni, L. He, and H. Yu, "Stochastic behavioral modeling and analysis for analog/mixed-signal circuits," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 32, no. 1, pp. 24–33, Jan. 2013.

[5] S. Weaver, B. Hershberg, P. Kurahashi, D. Knierim, and U.-K. Moon, "Stochastic flash analog-to-digital conversion," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 57, no. 11, pp. 2825–2833, Nov. 2010.

[6] S. Weaver, B. Hershberg, and U.-K. Moon, "Digitally synthesized stochastic flash ADC using only standard digital cells," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 61, no. 1, pp. 84–91, Jan. 2014.

[7] B. Xu, S. Li, N. Sun, and D. Z. Pan, "A scaling compatible, synthesis friendly VCO-based delta-sigma ADC design and synthesis methodology," in *Proc. IEEE/ACM Design Autom. Conf. (DAC)*, 2017, pp. 1–6.

[8] S. Brenna, A. Bonetti, A. Bonfanti, and A. L. Lacaita, "A tool for the assisted design of charge redistribution SAR ADCs," in *Proc. Design Autom. Test Eur. Conf. Exhibit (DATE)*, Grenoble, France, 2015, pp. 1265–1268.

[9] P. Nandi, H. Talukdar, D. Kumar, and A. K. G. Katakwar, "A novel approach to design SAR-ADC: Design partitioning method," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 35, no. 3, pp. 346–356, Mar. 2016.

[10] M. Ding *et al.*, "A hybrid design automation tool for SAR ADCs in IoT," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 26, no. 12, pp. 2853–2862, Dec. 2018.

[11] S. Ham *et al.*, "CMOS image sensor with analog gamma correction using nonlinear single-slope ADC," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, 2006, pp. 3578–3581.

[12] E. Reinhard, M. Adhikhmin, B. Gooch, and P. Shirley, "Color transfer between images," *IEEE Comput. Graph. Appl.*, vol. 21, no. 5, pp. 34–41, Jul./Aug. 2001.

[13] R. Likamwa, Y. Hou, Y. Gao, M. Polansky, and L. Zhong, "RedEye: Analog convnet image sensor architecture for continuous mobile vision," in *Proc. IEEE/ACM Int. Symp. Comput. Archit. (ISCA)*, 2016, pp. 255–266.

[14] P. Chi *et al.*, "PRIME: A novel processing-in-memory architecture for neural network computation in ReRAM-based main memory," in *Proc. IEEE/ACM Int. Symp. Comput. Archit. (ISCA)*, 2016, pp. 27–39.

[15] B. Li, P. Gu, Y. Shan, Y. Wang, Y. Chen, and H. Yang, "RRAM-based analog approximate computing," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 34, no. 12, pp. 1905–1917, Dec. 2015.

[16] W. Cao, X. He, A. Chakrabarti, and X. Zhang, "NeuADC: Neural network-inspired RRAM-based synthesizable analog-to-digital conversion with reconfigurable quantization support," in *Proc. Design Autom. Test Eur. Conf. Exhibit. (DATE)*, 2019, pp. 1477–1482.

[17] H. Zhu *et al.*, "CMOS image sensor data-readout method for convolutional operations with processing near sensor architecture," in *Proc. IEEE Asia–Pac. Conf. Circuits Syst. (APCCAS)*, Chengdu, China, 2018, pp. 528–531.

[18] A. Zou, H. Zhao, Y. Ma, and D. Li, "Analysis calculation and testing of rotary inductosyn angle measuring errors," in *Proc. 33rd Chin. Control Conf.*, Nanjing, China, 2014, pp. 8091–8096.

[19] A. Zou *et al.*, "Ivory: Early-stage design space exploration tool for integrated voltage regulators," in *Proc. 54th ACM/EDAC/IEEE Design Autom. Conf. (DAC)*, 2017, pp. 1–6.

[20] H.-S. P. Wong *et al.*, "Metal-oxide RRAM," *Proc. IEEE*, vol. 100, no. 6, pp. 1951–1970, Jun. 2012.

[21] J. Tapson and A. van Schaik, "An asynchronous parallel neuromorphic ADC architecture," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, 2012, pp. 2409–2412.

[22] Y. Xu, C. S. Thakur, T. J. Hamilton, J. Tapson, R. Wang, and A. van Schaik "A reconfigurable mixed-signal implementation of a neuromorphic ADC," in *Proc. IEEE Int. Biomed. Circuits Syst. Conf. (BioCAS)*, 2015, pp. 1–4.

[23] L. Gao *et al.*, "Digital-to-analog and analog-to-digital conversion with metal oxide memristors for ultra-low power computing," in *Proc. IEEE/ACM NANOARCH*, 2013, pp. 19–22.

[24] H. Li, P. Huang, B. Gao, B. Chen, X. Liu, and J. Keng, "A SPICE model of resistive random access memory for large-scale memory array simulation," *IEEE Electron. Device Lett.*, vol. 35, no. 2, pp. 211–213, Feb. 2014.

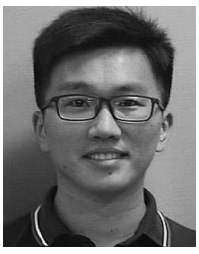
[25] X. Guan, S. Yu, and H.-S. P. Wong, "A SPICE compact model of metal oxide resistive switching memory with variations," *IEEE Electron. Device Lett.*, vol. 33, no. 10, pp. 1405–1407, Oct. 2012.

[26] P. Huang *et al.*, "A physics-based compact model of metal-oxide-based RRAM DC and AC operations," *IEEE Trans. Electron. Device*, vol. 60, no. 12, pp. 4090–4097, Dec. 2013.

[27] A. C. Torrezan, J. P. Strachan, G. Medeiros-Ribeiro, and R. S. Williams, "Sub-nanosecond switching of a tantalum oxide memristor," *Nanotechnology*, vol. 22, no. 48, 2011, Art. no. 485203.

[28] G. Khodabandehloo, M. Mirhassani, and M. Ahmadi, "Analog implementation of a novel resistive-type sigmoidal neuron," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 20, no. 4, pp. 750–754, Apr. 2012.

[29] B. Karlik and A. Vehbi, "Performance analysis of various activation functions in generalized MLP architectures of neural networks," *Int. J. Artif. Intell. Expert Syst.*, vol. 1, no. 4, pp. 111–122, 2011.
[30] S. R. Lee *et al.*, "Multi-level switching of triple-layered TaOx RRAM with excellent reliability for storage class memory," in *Proc. IEEE Symp. VLSI Technol. (VLSIT)*, 2012, pp. 71–72.
[31] S. Yu, B. Gao, Z. Fang, H. Yu, J. Kang, and H. S. Wong, "A low energy oxide-based electronic synaptic device for neuromorphic visual systems with tolerance to device variation," *Adv. Mater.*, vol. 25, no. 12, pp. 1774–1779, 2013.
[32] F. Bedeschi *et al.*, "A bipolar-selected phase change memory featuring multi-level cell storage," *IEEE J. Solid-State Circuits*, vol. 44, no. 1, pp. 217–227, Jan. 2009.
[33] K. Hornik, "Approximation capabilities of multilayer feedforward networks," *Neural Netw.*, vol. 4, no. 2, pp. 251–257, 1991.
[34] D. P. Kingma and J. L. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.
[35] W. Cao *et al.*, "A 40Gb/s 39mW 3-tap adaptive closed-loop decision feedback equalizer in 65nm CMOS," in *Proc. IEEE 58th Int. Midwest Symp. Circuits Syst. (MWSCAS)*, Fort Collins, CO, USA, 2015, pp. 1–4.
[36] W. Cao, Z. Wang, D. Li, F. Li, and Z. Wang, "A 40Gb/s adaptive equalizer with amplitude approaching technique in 65nm CMOS," in *Proc. IEEE Int. Conf. Electron Devices Solid-State Circuits (EDSSC)*, Singapore, 2015, pp. 451–454.
[37] M. Miyahara, I. Mano, M. Nakayama, K. Okada, and A. Matsuzawa, "A 2.2GS/s 7b 27.4mW time-based folding-flash ADC with resistively averaged voltage-to-time amplifiers," in *IEEE Dig. Tech. Papers Int. Solid-State Circuits Conf. (ISSCC)*, 2014, pp. 388–389.
[38] M. Prezioso, F. Merrikh-Bayat, B. D. Hoskins, G. C. Adam, K. K. Likharev, and D. B. Strukov, "Training and operation of an integrated neuromorphic network based on metal-oxide memristors," *Nature*, vol. 521, no. 7550, pp. 61–64, 2015.
[39] A. Kawahara *et al.*, "An 8 Mb multi-layered cross-point ReRAM macro with 443 MB/s write throughput," *IEEE J. Solid-State Circuits*, vol. 48, no. 1, pp. 178–185, Jan. 2013.
[40] S.-W. Chen, M.-H. Chang, W.-C. Hsieh, and W. Hwang, "Fully on-chip temperature, process, and voltage sensors," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, 2010, pp. 897–900.
[41] A. K. M. M. Islam and H. Ononera, "On-chip detection of process shift and process spread for post-silicon diagnosis and model-hardware correlation," *IEICE Trans. Inf. Syst.*, vol. E96.D, no. 9, pp. 1971–1979, 2013.
[42] A. Chen, "Utilizing the variability of resistive random access memory to implement reconfigurable physical unclonable functions," *IEEE Electron Device Lett.*, vol. 36, no. 2, pp. 138–140, Feb. 2015.
[43] B. Liu, H. Li, Y. Chen, X. Li, Q. Wu, and T. Huang, "Vortex: Variation-aware training for memristor X-bar," in *Proc. IEEE/ACM Design Autom. Conf. (DAC)*, San Francisco, CA, USA, 2015, pp. 1–6.
[44] L. Chen *et al.*, "Accelerator-friendly neural-network training: Learning variations and defects in RRAM crossbar," in *Proc. Design Autom. Test Eur. Conf. Exhibit. (DATE)*, Leuven, Belgium, 2017, pp. 19–24.
[45] F. Alibart, L. Gao, B. D. Hoskins, and D. B. Strukov, "High precision tuning of state for memristive devices by adaptable variation-tolerant algorithm," *Nanotechnology*, vol. 23, no. 7, 2012, Art. no. 075201.
[46] A. Fayyazi, M. Ansari, M. Kamal, A. Afzali-Kusha, and M. Pedram, "An ultra low-power memristive neuromorphic circuit for Internet of Things smart sensors," *IEEE Internet Things J.*, vol. 5, no. 2, pp. 1011–1022, Apr. 2018.



Weidong Cao (S'16) received the B.Eng. degree from Northwestern Polytechnical University, Xi'an, China, in 2013, and the M.Eng. degree from Tsinghua University, Beijing, China, in 2016, both in electrical engineering. He is currently pursuing the Ph.D. degree with the Preston M. Green Department of Electrical and Systems Engineering, Washington University in St. Louis, St. Louis, MO, USA.

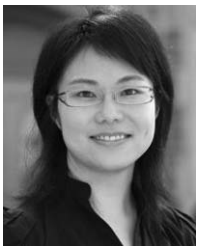
His current research interests include hardware accelerator, machine learning, in-memory computing/processing in-memory, and very large-scale integration design.

Xin He (M'17) received the B.Eng. degree in software engineering from Sichuan University, Chengdu, China, in 2011, and the Ph.D. degree in computer science from the Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China, in 2017.

He is currently a Post-Doctoral Research Fellow with the University of Michigan, Ann Arbor, MI, USA. His current research interests include computer architecture especially on application-specific acceleration, deep learning, and approximate computing.

Ayan Chakrabarti (M'07) received the B.Tech. and M.Tech. degrees in electrical engineering from the Indian Institute of Technology Madras, Chennai, India, in 2006, and the S.M. and Ph.D. degrees in engineering sciences from Harvard University, Cambridge, MA, USA, in 2008 and 2011, respectively.

He is an Assistant Professor with the Department of Computer Science and Engineering, Washington University in St. Louis, St. Louis, MO, USA. His current research interests include computer vision and machine learning, focusing on developing systems that can recover physical reconstructions and semantic descriptions of the world from visual measurements, for applications in robotics, autonomous vehicles, consumer photography, and graphics.



Xuan (Silvia) Zhang (S'08–M'15) received the B.Eng. degree in electrical engineering from Tsinghua University, Beijing, China, in 2006, and the M.S. and Ph.D. degrees in electrical and computer engineering from Cornell University, Ithaca, NY, USA, in 2009 and 2012, respectively.

She was a Post-Doctoral Fellow of Computer Science with Harvard University, Cambridge, MA, USA. She is an Assistant Professor with the Preston M. Green Department of Electrical and Systems Engineering, Washington University in St. Louis, St. Louis, MO, USA. Her current research interests include very large-scale integration, computer architecture, cyber-physical systems, adaptive power and resource management for autonomous systems, hardware/software co-design for machine learning and artificial intelligence, and efficient computation and security primitives in analog and mixed-signal domain.

Dr. Zhang was a recipient of the DATE Best Paper Award in 2019, and the ISLPED Design Contest Award in 2013, and has also been nominated for the Best Paper Award at DATE 2019 and DAC 2017.