MNSIM: Simulation Platform for Memristor-based Neuromorphic Computing System

Lixue Xia, *Student Member*, *IEEE*, Boxun Li, *Student Member*, *IEEE*, Tianqi Tang, Peng Gu, *Student Member*, *IEEE*, Pai-Yu Chen, *Student Member*, *IEEE*, Shimeng Yu, *Member*, *IEEE*, Yu Cao, *Fellow*, *IEEE*, Yu Wang, *Senior Member*, *IEEE*, Yuan Xie, *Fellow*, *IEEE*, Huazhong Yang, *Senior Member*, *IEEE*

***Abstract*—Memristor-based computation provides a promising solution to boost the power efficiency of the neuromorphic computing system. However, a behavior-level memristor-based neuromorphic computing simulator, which can model the performance and realize an early-stage design space exploration, is still missing. In this paper, we propose a simulation platform for the memristor-based neuromorphic system, called MNSIM. A hierarchical structure for memristor-based neuromorphic computing accelerator is proposed to provides flexible interfaces for customization. A detailed reference design is provided for large-scale applications. A behavior-level computing accuracy model is incorporated to evaluate the computing error rate affected by interconnect lines and non-ideal device factors. Experimental results show that MNSIM achieves over 7000 times speed-up than SPICE simulation. MNSIM can optimize the design and estimate the trade-off relationships among different performance metrics for users.**

I. Introduction

The neuromorphic algorithms have shown great performance in many fields including vision, speech, and other intelligent processing tasks [1]. The neuromorphic algorithms cost much more memory and computing resources than traditional methods, which necessitates the energy-efficient design in modern computing systems [2]. However, it has become more and more difficult to achieve substantial power efficiency gains through the scaling down of traditional CMOS technique [**?**], [3], and the "memory wall" bottleneck affects the efficiency of von Neumann architecture [4].

The innovation of memristor and memristor-based computing system provides a promising solution to boost the power efficiency of neuromorphic computation. The inherent analog resistant characteristic provides an alternative to the von Neumann architecture by processing the computation in memory [5]. Therefore, several memristor-crossbar-based neuromorphic systems and accelerators have been developed to improve energy efficiency significantly [6], [7].

This work was supported by 973 project 2013CB329000, and National Natural Science Foundation of China (No.61622403, 61373026), National Key Research and Development Program of China (No. 2017YFA0207600), Joint fund of Equipment pre-Research and Ministry of Education (No. 6141A02022608), and Tsinghua University Initiative Scientific Research Program.

L. Xia, B. Li, T. Tang, Y. Wang, H. Yang are with the Department of Electronic Engineering, Tsinghua National Laboratory for Information Science and Technology, Tsinghua University, Beijing 100084, China (e-mail: yu-wang@tsinghua.edu.cn).

P. Chen, S. Yu, Y. Cao are with the School of Electrical, Computer and Energy Engineering, Arizona State University, Arizona, USA.

P. Gu, Y. Xie are with the Department of Electrical and Computer Engineering, University of California at Santa Barbara, California, USA.

A memristor-based neuromorphic accelerator performs neuromorphic computation in high energy efficiency, which is the most important part of a memristor-based neuromorphic computing system. For large-scale applications on memristor, a large number of different factors can affect the performance. Therefore, accurate early-stage simulation is needed to estimate and optimize the performance of the design. However, none of the existing system simulation platforms completely supports the simulation of a memristor-based neuromorphic computing system and accelerator. Traditional architectural simulators like GEM5 [8] cannot support memristor devices and memristor-based computing structures. NVSim [9] and NVMain [10] are memory-oriented simulators where the peripheral circuit structure of NVSim/NVMain is fixed for memory simulation. Therefore, circuit-level simulators such as SPICE and NVMspice [11] are used to simulate the memristor-based accelerator. However, the simulation time increases dramatically when the scale of the network gets larger. As a result, a fast simulation platform specific to memristor-based neuromorphic accelerator with an accurate behavior-level model is highly demanded.

Several challenges need to be addressed when developing a behavior-level simulation platform for the memristor-based neuromorphic accelerator. First, since the detailed circuit designs of memristor-based neuromorphic accelerators vary a lot, a simulation platform needs to integrate a flexible architecture for the accelerator to support the various designs. Second, computing accuracy is the main metric that needs to be estimated in a memristor-based neuromorphic accelerator, but a behavior-level estimation model is still missing.

This work proposes MNSIM, a simulation platform that simulates the memristor-based neuromorphic accelerator from the behavior level. The main contributions of MNSIM are as follows:

1) A hierarchical structure for memristor-based neuromorphic computing accelerator is proposed to support various algorithms with the same structure abstraction, and all the design parameters are classified into the three levels in the proposed hierarchical structure.
2) MNSIM provides both a reference design for large-scale neuromorphic accelerator and flexible interfaces for users from multiple levels to customize their designs.
3) A behavior-level computing accuracy model is proposed

to estimate the average and worst cases with high speed, which accelerates the estimation more than 7000× compared with SPICE.

4) MNSIM can explore the design space of memristor-based neuromorphic computing accelerators for design optimization. The simulation results can guide the design of memristor-based neuromorphic computing systems at an early design stage.

II. Preliminaries

A. Memristor and Memristor-based Computing Structure

A memristor cell is a passive two-port element with variable resistance states. There are multiple kinds of devices which can be used as memristor cells, such as Resistive Random Access Memory (RRAM), Phase Change Memory (PCM), etc. Multiple memristor cells can be used to build the crossbar structure. If we store each value of a "matrix" by the conductance of the memristor cell ($g_{k,j}$) and input the "vector" signals through variable voltages, the memristor crossbar can perform analog matrix-vector multiplication, as shown in Fig. 1(e). The relationship between the input voltage vector and output voltage vector can be expressed as follows [12]:

$$\begin{bmatrix} v_{\text{out},1} \\ \vdots \\ v_{\text{out},M} \end{bmatrix} = \begin{bmatrix} c_{1,1} & \cdots & c_{1,N} \\ \vdots & \ddots & \vdots \\ c_{M,1} & \cdots & c_{M,N} \end{bmatrix} \begin{bmatrix} v_{\text{in},1} \\ \vdots \\ v_{\text{in},N} \end{bmatrix} \tag{1}$$

where $v_{\text{in},j}$ denotes the jth element of input voltage vector ($j = 1, 2, ..., N$), $v_{\text{out},k}$ denotes the kth element of output voltage vector ($k = 1, 2, ..., N$), and $c_{k,j}$ is the weight matrix for multiplication. The matrix data can be represented by the conductances of the memristor cells and the conductance of the load resistor (g_s) as [12]:

$$c_{k,j} = \frac{g_{k,j}}{g_s + \sum\limits_{l=1}^{N} g_{k,l}} \tag{2}$$

Since the memristor cells also serve as memory devices in this structure, the computation and memory are merged during operation. Therefore, the matrix-vector multiplications can be efficiently processed by memristor crossbars [6].

B. Memristor-based Neuromorphic Computing

Since matrix-vector multiplication operations dominate the majority of the computation work of neuromorphic algorithms [13], [14], researchers have proposed various memristor-based neuromorphic computing architectures. However, not all algorithms can be efficiently implemented by memristor-based computing structure. In this paper, three classes of neuromorphic algorithm implementations are explored based on memristor computing platforms.

1) Memristor-based DNNs: In a layer of DNN, each output neuron is connected to all the input neurons of this layer, which is called a fully-connected layer. The function to calculate output signals in a fully-connected layer with N input neurons and M output neurons is:

$$\text{output}_k = f(\sum \text{input}_j * \text{weight}_{k,j} + \text{bias}_k) \tag{3}$$

where output_k denotes the calculated output signal of the kth output neuron ($k = 1, 2, ..., M$), input_j denotes the input signal from the jth input neuron ($j = 1, 2, ..., N$) , $\text{weight}_{k,j}$ denotes the weight connecting the jth input neuron and the kth output neuron, and bias_k is the bias signal of the kth output neuron. f is a non-linear function to introduce non-linear classification ability into the algorithm, such as sigmoid function and the rectified linear unit (ReLU) function. Equ. (3) can be rewritten into a matrix version:

$$\textbf{Output}_{M\times 1} = f(\textbf{Weight}_{M\times N}\textbf{Input}_{N\times 1} + \textbf{Bias}_{M\times 1}) \tag{4}$$

As Equ. (4) shows, the main function of fully-connected neuromorphic algorithms is based on matrix-vector multiplication, namely $\textbf{Weight}_{M\times N} \times \textbf{Input}_{N\times 1}$. The matrix-vector multiplication function is usually called **synapse** function in neuromorphic algorithms, and the non-linear function is called **neuron** function. The synapse function can be efficiently implemented by memristor crossbars, and the neuron function is implemented by peripheral modules [15]. When processing a well-trained neuromorphic network, the **Input** vector changes in different samples (e.g., different pictures for image classification), but the **Weight** matrix remains the same. Therefore, once mapping a well-trained network onto memristor-based computing circuit, the resistant states of memristor cells will not need to be changed during neuromorphic computing operations. This characteristic avoids the high-writing-cost problem [6] and the endurance limitation [16] of memristor devices. As a result, memristor crossbar provides a promising solution to boost the energy efficiency of fully-connected neuromorphic networks [17].

Researchers have proposed several kinds of memristor-based DNN accelerators. Li [12] proposed a memristor-based approximate computing system, and Hu [17] used the memristor-based computing structure to accomplish the recall function in an auto-associative neural network. When the network scale increases, multiple memristor crossbars need to work together for a large weight matrix. Hu [18] and Liu [19] have provided the peripheral circuits to merge the results of multiple memristor crossbars.

2) Memristor-based SNNs: SNN is a brain-inspired algorithm where the information is encoded by the precise timing of spikes [20]. Since the synapse function is also matrix-vector multiplication, researchers have used memristor crossbars to implement the synapse function of SNN [21]. Some researchers also use the dynamic synaptic properties of memristor devices to perform the learning and selecting functions [22]. This inspiring work has shown the potential of further improving the efficiency by memristor devices, but a proper model for the dynamic synaptic property is still missing. Therefore, in the following sections of this paper, we only focus on the SNNs that use each memristor cell to store a fixed weight. Since the function of a SNN layer can also be described by Equ. (4), we regard both DNN and SNN as fully-connected networks in the following sections.

3) Memristor-based CNNs: CNN has shown great performance in computer vision field [14]. In CNN, convolutional (Conv) layers are cascaded before fully-connected layers to extract the local features. The primary function of a Conv layer

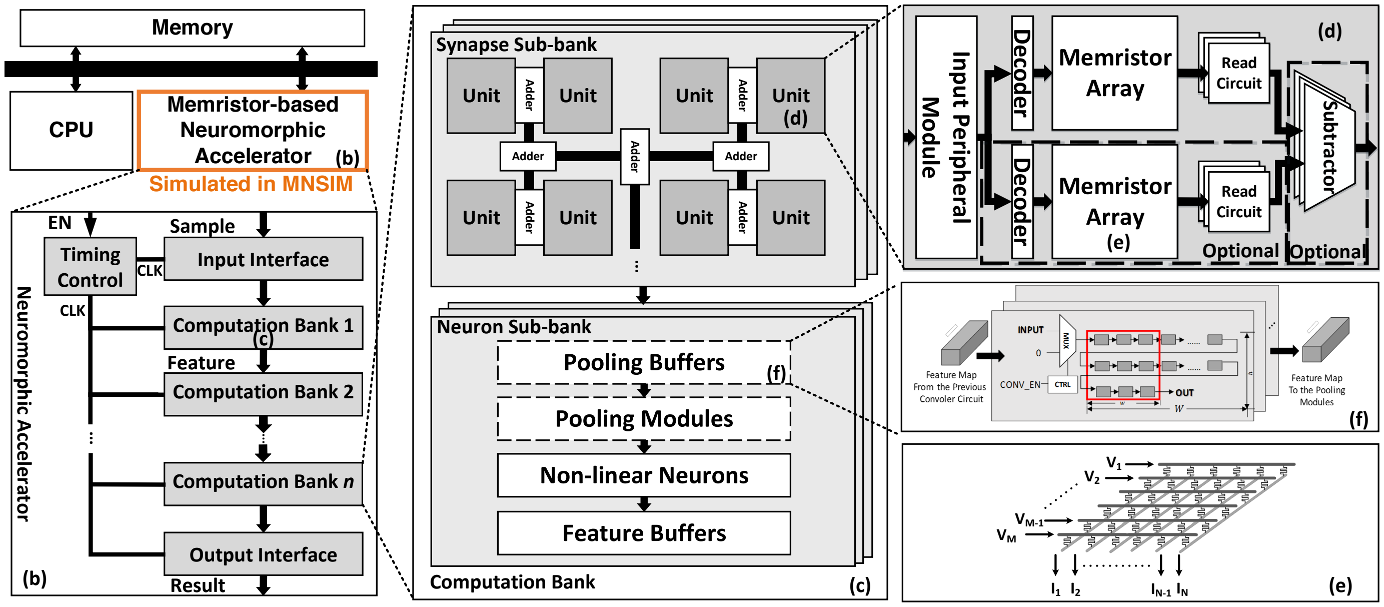


Fig. 1: (a) The architecture of entire Memristor-based Neuromorphic Computing System. MNSIM focuses on the simulation for the Memristor-based Neuromorphic Accelerator. (b) Hierarchical Structure of Memristor-based Neuromorphic Accelerator. (c) Computation Bank that consists of Synapse Sub-banks and Neron Sub-banks. (d) Computation Unit, where the circuits in dotted blocks are optional to implement signed weight. (e) Memristor Crossbar (f) Line Buffer structure.

is the convolution (Conv) kernel, which can also be regarded as vector-vector multiplication [23]. Since multiple kernels in the same layer share the input vectors, multiple kernels can be regarded as matrix-vector multiplication. Therefore, all layers of CNN can be implemented in memristor crossbars with high energy efficiency [7], [24]. Memristor-based convolution kernels has already been validated by memristor chips [25].

Since CNNs are not fully-connected, the detailed circuit designs of memristor-based CNN accelerators are different from the others. However, we find that the overall architectures are similar if we abstract the circuits into higher-level modules, as described in Section III.

C. Difference between Memristor-based Non-volatile Memory and Memristor-based Computing Structure

Researchers have fabricated non-volatile memories using memristor [26], but these circuits cannot support memristor-based computation. First, in memory operations, i.e., the READ and WRITE operations, only one cell is selected in each crossbar [6]. However, when processing matrix-vector multiplication, all the memristor cells in a crossbar need to be selected to entirely utilize the parallelism of crossbar structure, as shown in Fig. 1(f). Therefore, the cell selection scheme and the corresponding control circuits need to be adjusted. Second, peripheral computing modules are necessary to support the entire function of neuromorphic computation.

D. Related Work about Simulation Platform

GEM5 [8] is widely used in system designs based on CMOS technology, but it does not support the memristor device and memristor-based computing structure. NVSim [9] and NVMain [10] are simulators for non-volatile memory using memristor cells or CMOS technologies. However, the peripheral circuit structure is fixed for memory design, so NVSim and NVMain do not support the simulation of computing structure. NVMspice [11] simulates the state of memristor from circuit level, which requires that the memristor cell's I-V characteristic must be linear or quadratic. Since practical memristor cells' characteristics are far from these ideal curves [17], NVMspice cannot be used to simulate the performance of a real system. Therefore, researchers currently use the circuit-level simulator like SPICE to optimize the circuit design, but the optimization of a large-scale application will take months to simulate a large number of designs iteratively. Therefore, a high-speed simulation platform that supports the simulation of a memristor-based neuromorphic accelerator is highly demanded.

III. Hierarchical Structure of Memristor-based Neuromorphic Computing Accelerator

The memristor-based neuromorphic computing accelerator can be regarded as a piece of equipment that is connected to CPU through the bus, as shown in Fig. 1(a). Since the simulation of CPU, memory, and other I/O devices are well settled by existing architectural simulators like GEM5 [8], MNSIM only simulates the performance of memristor-based neuromorphic accelerator.

The design of a memristor-based neuromorphic accelerator involving both the architecture-level optimization (e.g., the connection of crossbars) and the circuit-level optimization (e.g., choosing suitable read circuits). A behavior-level simulation platform needs to support the modifications of memristor-based neuromorphic accelerator from different levels, without changing the fundamental simulation flow or re-writing the source code. Therefore, it is necessary to abstract the design differences into quantifiable parameters and decouple them into multiple design levels. To solve this problem, we propose a hierarchical structure abstraction for memristor-based neuromorphic accelerators.

The neuromorphic algorithm consists of multiple cascaded network layers with similar functions. Therefore, the neuromorphic accelerator can be divided into multiple cascaded modules, where each module represents the operation of a network layer. This module is named **Computation Bank** in MNSIM.

The scale of a network layer can be larger than the maximum size of a memristor crossbar [7]. Therefore, to process large-scale neuromorphic application, the computation bank must contain multiple memristor crossbars. The crossbars are connected with some necessary peripheral circuits, which constitutes as multiple individual units. MNSIM integrates these individual units as **Computation Unit**s.

As a result, most of the circuit designs of memristor-based neuromorphic accelerators can be represented by the hierarchical structure containing three levels: **Accelerator** level, **Computation Bank** level, and **Computation Unit** level. The hierarchical structure is shown in Fig. 1. Based on this abstraction, MNSIM provides flexible customization interfaces, and the architecture remains unchanged. As shown in Table I, users can configure the design from different levels for different neuromorphic accelerators. In addition, MNSIM assumes that all the weight matrices can be entirely stored in multiple memristor crossbars. This is because that high integration is an advantage of memristor-based structure, and storing all weights without repeatedly writing can better improve the energy efficiency of memristor-based computing structure [7].

A detailed reference design of each level is provided below, and users can modify the detailed structure through customization interfaces, as described in Section III.E.

A. Level-1: Accelerator

As shown in Fig. 1(b), the accelerator contains not only the cascaded computation banks but also some modules that support the functions for the whole accelerator like the accelerator interfaces. The accelerator-level design mainly focuses on determining the number of computation banks and the number of ports for the interfaces. As Table I shows, the accelerator level contains two main parameters: $Interface\_Number$ and $Network\_Depth$.

The $Network\_Depth$, namely the number of neuromorphic layers in the network, is determined by the target application. The $Network\_Depth$ determines the number of computation banks contained in the accelerator. In CNN, the definition of a "layer" varies in different work [13], [27], including ReLU layers, pooling layers, buffer layers, etc. However, as mentioned in Section II.B, only the Conv kernels can be efficiently computed by memristor crossbars. Therefore, MNSIM only regards a layer that contains Conv kernels or fully-connected weights as a neuromorphic layer to be processed in one computation bank. In this way, all the subsequent functions after Conv kernels in a CNN is regarded as the peripheral functions in a computation bank. For example, the widely-used CaffeNet [27] is regarded as a 7-layer CNN and contains seven computation banks.

In memristor-based computing, all the inputs of a crossbar need to be well prepared before the computation cycle. Since the size of input samples and output computation results can be larger than the number of wires in the data bus, we need interface modules to buffer the input and output data of accelerator. The input module sends the data to the first computation bank after a sample has been completely received from the bus. In this way, the input module uses limited $Interface\_Number$[1] input lines to accomplish the transmission of all the input data in a sample and keeps the fully-parallel function of memristor crossbars. Similarly, an output module is cascaded after the final computation bank to send the output results in multiple cycles.

B. Level-2: Computation Bank

A computation bank processes the computation of one neuromorphic layer. Each computation bank consists of multiple **Computation Unit**s, an **Adder Tree** merging the results for a large neuromorphic layer, a **Peripheral Module** for subsequent functions like non-linear function, the **Pooling Buffer** to process the spatial pooling in CNN, and the final **Output Buffer**. Computation bank is further divided into multiple synapse sub-banks and multiple neuron sub-banks according to the corresponding synapse function and neuron function in the neuromorphic algorithms, where multiple computation units using the same inputs belong to a synapse sub-bank. Since the digital signals have better tolerance for noise when merging the outputs of multiple units, MNSIM uses digital input/output signals for computation units and the other modules as a reference design. If users want to use analog communication, they can move the read circuits from units to peripheral modules to process the simulation.

1) Computation Unit: When using multiple crossbars to implement the matrix-vector multiplication, a large matrix is divided into multiple blocks. The results of multiple small matrix-vector multiplications of a row need to be merged by an addition operation as:

$$\vec{V}_{\text{out},k} = \sum_{j=1}^{N} W_{k,j} \vec{V}_{\text{in},j} \tag{5}$$

where $\vec{V}_{\text{in},k}$ is the sub-vector of input data in the kth row (e.g., $\vec{V}_{\text{in},1} = (v_{\text{in},1}, v_{\text{in},2}, ..., v_{\text{in},s})$ if the $Crossbar\_Size$ is s), $\vec{V}_{\text{out},j}$ is the sub-vector of output data in the jth row, and $W_{k,j}$ is the sub-matrix in the kth row and jth column.

In the proposed hierarchical structure, each unit processes the matrix-vector multiplication of a sub-matrix and a sub-vector. The detailed structure of a computing unit will be discussed in Section III.C.

2) Adder Tree: As Equ. (5) shows, results from multiple units are added together. We use an adder tree structure as shown in Fig. 1(c), where the results of two neighboring units are added first, and then merged by a binary tree structure.

Since the number of resistance levels in one memristor cell cannot support a high-precision weight, researchers store the lower bits and higher bits of a weight matrix into multiple memristor crossbars and merged the output results. The merging function can also be implemented by the adder tree, but the shifters need to be added [6].

3) Pooling Module and Pooling Buffer: The spatial pooling function in CNN chooses the maximum value of the neighboring $k \times k$ results in a matrix. Therefore, a pooling module is cascaded after the adder tree in memristor-based CNNs.

However, only a maximum function is not enough for memristor-based design. Since the pooling inputs are obtained in multiple cycles, the temporary results before pooling need to be buffered. Since the living period of the buffered data is much smaller than the total data amount, researchers have proposed a pooling line-buffer inspired by the convolutional line-buffer used in a FPGA-based CNN accelerator [28], as Fig. 1(f) shows. In each iteration, a new result coming from the adder tree is buffered into the head, and all the other data shift for one register. As a result, the data in the red block are just the inputs for the next pooling function.

4) Non-linear Neuron Module: Equ. (3) has shown that a non-linear neuron function at the end of a neuromorphic layer is needed to provide non-linear classification ability. The non-linear neuron function cannot be separately operated before the linear adding operation in Equ. (5). Therefore, the neuron functions can only be implemented outside the units, and operate after the adder tree. Considering that all the existed non-linear functions used in CNNs are monotone increasing functions, the pooling function can be processed before non-linear neurons to reduce processing times of neurons. MNSIM cascades the neuron module after the adder tree in fully-connected NNs, or after the pooling module in CNNs. The reference design of neuron module is the sigmoid function for DNN, the integrate and fire function for SNN, and the ReLU function for CNN.

5) Output Buffer: In fully-connected layers, the output buffer size is same to the number of neurons C_{out}. Therefore, the output buffer consists of C_{out} registers and each register is connected to a neuron module through a fixed wire. In the cascaded Conv layers of CNN, not all the outputs are required simultaneously for next Conv layer. Similar to the pooling buffer, MNSIM integrates a simplified line buffer, as Fig. 1 shows. For the ith Conv layer whose output feature map size is $W^{i+1} \times H^{i+1} \times C_{\text{in}}^{i+1}$ the output buffer contains C_{in}^{i+1} separate line buffers. If the size of convolution kernel in the $(i+1)$th layer is $w^{i+1} \times h^{i+1}$, the length of a single line buffer L_{out}^{i} is:

$$L_{\text{out}} = W^{i+1} \times (h^{i+1} - 1) + w^{i+1} \tag{6}$$

Therefore, in the $(i+1)$th layer, the data in the blocked registers are used for convolution function, and the processing of Conv layers are pipelined through the data flowing in line buffers.

C. Level-3: Computation Unit

A computation unit consists of four main modules: memristor crossbar, address decoder, read circuit, and input peripheral circuit. The reference structure of computation unit provided in MNSIM is shown in Fig. 1(d).

1) Memristor Crossbar: accomplishes the memory and computation functions of a neuromorphic accelerator. Since the resistance of memristor devices can only be positive, two memristor cells are needed to represent one signed weight. There are two methods: (1) a unit needs to contain two crossbars and uses the difference value of the corresponding cells to reflect one signed weight [17], as shown in Fig. 1(d); (2) both positive and negative values are stored in the same crossbar, and two outputs from different columns are subtracted. In MNSIM, the polarity of network weights and the mapping method of the signed weights are configurable. Therefore, the second memristor crossbar and the subtractors in the computation unit are optional, and the $Crossbar\_Size$ can be adjusted for design space exploration.

2) Address Decoder: is the module to select specific column/row through a transfer gate. In READ and WRITE operation, two decoders are needed for each crossbar to select the specific row and column. A computation-oriented crossbar decoder is designed to select all the input ports of a crossbar, which is introduced in Section V.B. The detailed decoder circuit is determined by $Crossbar\_Size$.

3) Input Peripheral Circuit: contains DACs and transfer gates as switches to generate the input signals. In computing phase, all the inputs should be provided in the same cycle to fully utilize the parallelism of memristor-crossbar-based computation. Since the crossbar size and the matrix size may not match, the input throughput of a crossbar is the smaller value between the number of rows in the weight matrix and the number of rows in memristor crossbar.

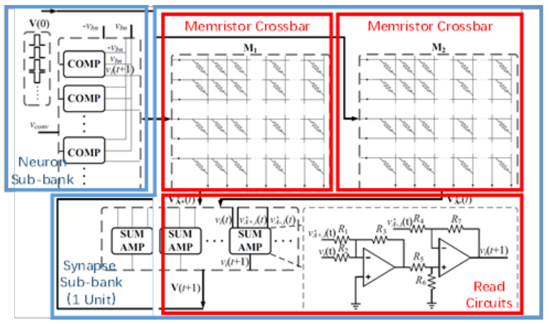
4) Read Circuits: are ADCs or multilevel Sensing Amplifiers (SAs) and the extra subtractors to merge the signals come from two crossbars. To reduce area and power consumption, some researchers propose that each crossbar only computes p columns in one cycle, and sequentially compute multiple cycles to obtain the entire results of a crossbar. MNSIM uses the parallelism degree p as a variable for optimization. A larger p leads to higher output throughput but higher area overhead. A control module is used to route the crossbar output to sensing circuit through MUXs, which can be implemented by a digital counter.

D. Instruction Set and Controller

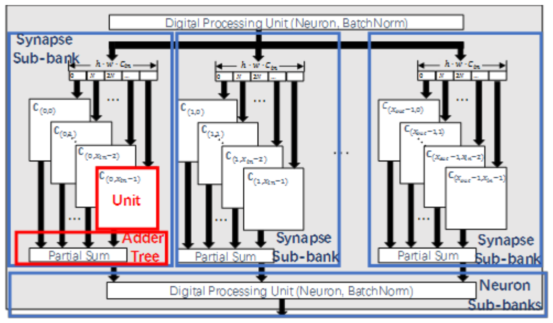
The instruction sets vary a lot in different memristor-based neuromorphic computing accelerators. For example, a general-deep-learning accelerator can support a fine-grained instruction set that is similar to Cambricon [29], but an application-specific accelerator may only support three basic instructions: WRITE, READ, and COMPUTE. As a simulator platform, MNSIM first supports the design using three basic instructions. If the designer uses other customized instructions to support improved functions, they can design their own instruction sets and integrate corresponding control modules. This customization will not change the simulation flow of MNSIM.

E. Customized Design

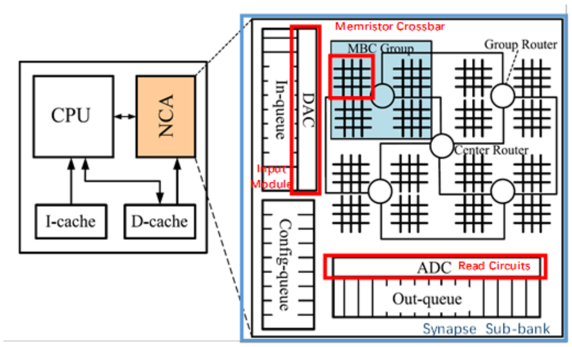
MNSIM provides a reference design of each circuit module based on the above hierarchical structure. If other models



(a)



(b)



(c)

Fig. 2: Examples of the neuromorphic circuit structures that MNSIM supports: (a) Small Network from [17], (b) Large Network on Memristor from [28], and (c) Large Network with CPU from [19].

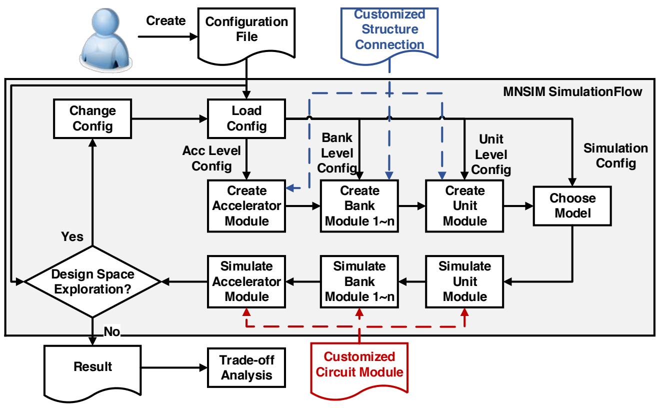


Fig. 3: The software flow of MNSIM.

TABLE I: The Configuration List of MNSIM.

Inputs	Level	Default Value	Comment
Network_Depth	Accelerator	-	Layers of Application
Interface_Number	Accelerator	[128,128]	Input and Output I/O Amount
Network_Type	Bank	ANN	Algorithm Type
Network_Scale	Bank	-	Scale of Each Layer
Crossbar_Size	Bank	128	
Pooling_Size	Bank	2	
Spacial_Size	Bank	1	
Weight_Polarity	Unit	2	
CMOS_Tech	Unit	90nm	
Cell_Type	Unit	1T1R	1T1R/0T1R
Memristor_Model	Unit	RRAM	
Interconnect_Tech	Unit	28nm	
Parallelism_Degree	Unit	0	0 means All Parallel
Resistance_Range	Unit	[500 500k]	Min/Max Memrstor Resistance

and topologies of memristor-based computing need to be simulated, users can customize the modules and connection structures from different levels.

1) Mapping circuits onto the reference design of MNSIM: The reference design of MNSIM directly supports the simulation of a large number of existing memristor-based neuromorphic accelerators [12], [17], [23], [25]. When implementing small scale applications, we only need one computation unit in each computation bank, as shown in Fig. 2(a). For large networks, each computation bank contains multiple computing units, as shown in Fig. 2(b).

2) Customizing structure connection: Users can customize the connection of modules on multiple levels. For a computation-bank-level example, if the users want to build an accelerator which only consists of multiple reconfigurable computation units [6], they can distribute the peripheral modules like the adders into computation units. For a computation-unit-level example, the structure proposed in [24], [30] eliminates the DACs and ADCs (or multilevel SAs) around each crossbar. The users can remove the corresponding modules of computation unit to simulate this structure.

3) Customizing module: Users can change the performance model of a circuit module in the reference design. If users want to introduce new modules, like mapping a new kind of neuromorphic algorithms onto the memristor-based structure, they only need to integrate the performance models into MNSIM. For example, Fig. 2(c) shows the design in [19]. The memristor-based accelerators only accomplish the computation of synapse function, and other functions are processed in a CPU. Therefore, the accelerator part only contains synapse sub-bank where the adder tree is replaced by an analog router, and the buffer design is modified. To simulate this work, the user needs to provide the power, latency, area, and accuracy loss models of the new modules and adds them to the simulation function of synapse sub-bank.

4) Cooperate with other simulator: NVSim [9] is a powerful simulator for memristor-based non-volatile memory. We provide an interface for each computation-oriented module (i.e. sigmoid circuit) to be compatible with NVSim. Users can easily introduce some NVSim results into MNSIM; or use MNSIM results in NVSim by adding the circuit models.

IV. Simulation Platform

A. *Software flow and Inputs of MNSIM*

Based on the hierarchical structure described in Section III, users provide a configuration file which contains the module choices and the design parameters. The details about configuration file are shown in Table I. MNSIM generates modules of Accelerator level, Computation Bank level, and Computation Unit level using the configuration file. As shown in Fig. 3, MNSIM first generates a memristor-based neuromorphic accelerator module which contains the I/O interface and multiple cascaded computation banks. After that, MNSIM generates the computation bank modules and computation unit modules recursively. The performance of each module is simulated using the models described in Section V and the technique configurations. The simulation works from computation unit level back to accelerator level, as shown in Fig. 3.

As a behavior-level simulation platform, MNSIM adds the performance of the lower-level modules together to obtain the performance of a higher-level module. For latency estimation, MNSIM uses the worst case to evaluate execution time for three reasons. First, memristor-based neuromorphic computing works in a cascaded way, as Fig. 1(c) shows. Therefore, the critical path is determined by the structure connection instead of the input signals. Second, most memristor-based multi-layer accelerators use pipelined design [28], so the execution time is determined by the worst-case delay among layers. Third, neuromorphic computing is used for recognition of inputs that are not predictable; hence, users mainly concern the recognition speed in the worst case.

MNSIM estimates the area, power, latency, and computing accuracy based on the CMOS technology data and estimation models discussed in the following subsections. If users do not determine all configurations, MNSIM will give the optimal design for each performance with design details. If users still want to perform a circuit-level simulation with specific weight matrices and input vectors, MNSIM can generate the netlist file for circuit-level simulators like SPICE.

MNSIM provides friendly interfaces for customization. If users want to change the connection relationship of modules, they only need to modify the module generation function of MNSIM, as the blue dotted line shows in Fig. 3. If users want to add new circuit modules, they can add the performance model into the simulation functions as the red dotted line shows in Fig. 3

B. *Limitations of MNSIM*

First, MNSIM is specific for memristor-based neuromorphic computing where the memristor cells are used as fixed weights. Some researchers focus on the dynamic properties of the memristor and use memristors as oscillators or other dynamic modules [31], which cannot be supported by MNSIM.

Second, as a behavior-level simulation platform, MNSIM obtains simulation speed-up at the cost of simulation accuracy, which provides an early-stage simulation and supports the user to compare different designs. However, after choosing or reducing the search region of some key design parameters, designers are still suggested to perform a circuit-level simulation using the detailed model and technology files before fabrication.

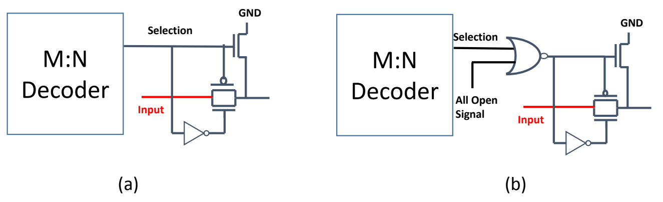


Fig. 4: (a) Memory-oriented Decoder. (b) Computation-oriented Decoder to select all cells. A NOR gate is added.

V. Area, Power, and Latency Models

A. *Memristor Crossbar*

The crossbar structure for computation is the same as the structure in a memristor-based memory structure, MNSIM uses the existing area model of memristor-based memory to estimate the crossbar area. The area estimation models of MOS-accessed cell and cross-point cell are [32]:

$$AREA_{\text{MOS-accessed}} = 3(W/L+1)F^2 \tag{7}$$

$$AREA_{\text{cross-point}} = 4F^2 \tag{8}$$

where W/L is the technology parameter of the transistor in each cell, and F is the size of memristor technology node.

Since all cells are used in computing, the power consumption is larger than that of a memory-oriented circuit. MNSIM uses the harmonic mean of minimum and maximum resistance of memristor to replace all cells' resistance values as the average case estimation.

B. *Decoder*

The traditional decoder used in memory is an address selector controlling the transfer gate between the input signal and crossbar, as shown in Fig. 4(a). MNSIM modifies this decoder into a computation-oriented decoder shown in Fig. 4(b). A NOR gate is placed between the address decoder and the transfer gate, and a control signal is connected to the NOR gate. When processing computation, the control signal is at high voltage and turns on all transfer gates through the NOR gate.

C. *ADC*

Researchers have shown that ADC circuits take about half of the area and energy consumptions in memristor-based DNNs and CNNs [24]. There are two main parameters when choosing ADC: the precision and frequency. The precision of ADC can be calculated by the crossbar input data precision, weight precision, and the crossbar size [7], and MNSIM also uses this method to determine the precision of ADC. Since neuromorphic computing is a kind of approximate computing, the precision of most CNNs can be quantized into 8-bit fixed-point value [14], which means the precision of ADC can also be 8-bit. Therefore, the precision of ADC can be directly configured according to the algorithm requirements. For frequency, there is a trade-off between speed and hardware overhead of ADC. A

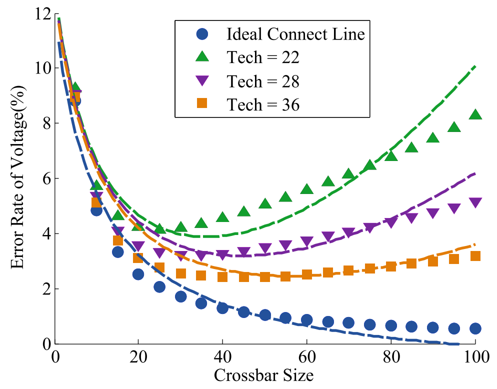


Fig. 5: The error rate fit curves of output voltages with different crossbar sizes and interconnect technology nodes. The scattered points are simulated by SPICE, and the lines reflect the proposed estimation model.

widely-used assumption is that the frequency of ADC should match the speed of memristor-based computing structure. The latency of memristor-based non-volatile memory has been reduced to 10 $\sim$ 100ns [7], [33], which motivates us to choose an ADC whose frequency is higher a 10MHz.

MNSIM uses a variable-level SA [34] with 50MHz frequency as the reference ADC design [6]. There are a lot of kinds of ADCs with different performances [35], and the performance models of some popular ADC designs have been integrated into MNSIM. Users can also integrate the area, power, and latency performance model of a customized ADC into MNSIM.

D. Other Modules

MNSIM provides a reference transistor-level design and uses the technology parameters from CACTI [36], NVSim [9], Predictive Technology Model (PTM) [37], [38], and other technology documents to estimate the parameters of transistor-based modules [**?**].

For a customized module, the user can provide a detailed transistor-level design, or directly provide the behavior-level dynamic power, static power, latency, and area performance of the module.

VI. Behavior-Level Computing Accuracy Model

Computing accuracy is an important characteristic for a computing accelerator. For memristor-based fixed-point computation, the computing error can be divided into two parts: 1) the error generated by data quantization, and 2) the error caused by memristor-based analog computation. To fairly evaluate the performance of memristor-based hardware structure, MNSIM uses the second part of error as the definition of computation error, and therefore the ideal computation result is based on the fixed-point algorithm.

Traditional circuit-level simulators solve a large number of non-linear Kirchhoff equations to obtain the accuracy loss. For an $M \times N$ crossbar, more than $[MN + M(N-1)]$ voltage variables (the voltages of the input node and output node of each cell) and $3MN$ currents need to be solved. Moreover, since memristors are devices with non-linear V-I characteristics [39], the equations are not linear. Consequently, simulation speed of circuit-level simulator is limited by the high cost of solving non-linear equations.

We propose three approximation steps to obtain a behavior-level computing accuracy model: (1) decoupling the influence of non-linear V-I characteristic to simplify equations into linear format, (2) ignoring the capacitance and inductance of interconnect lines to simplify the crossbar circuits into multiple series resistor sets, and (3) using the average case and the worst case to reduce the model complexity.

A. Decoupling the Influence of Non-linear V-I Characteristics

When analyzing analog circuits, we first find the DC operating point of each device according to some approximations, and then add the influence of small-signal parameters onto the operating point. Inspired by this method, we first regard each memristor cell as a variable resistor with a linear V-I characteristic in each resistance state, where the resistance of each cell R_{idl} is determined according to Equ. (2). In this way, the equations are transformed into linear equations, and the ideal operating voltages are obtained. After that, the influence of non-linear V-I characteristic is added back to calculate the actual resistance R_{act} and the current of each memristor cell at the given operating voltage.

B. Ignoring the Capacitance and Inductance of Interconnect Lines

In a circuit-level simulator, the interconnect line model contains a resistor r, a cascaded inductor, and a bridged capacitor. Since the inductances and capacitances of interconnect lines have little influence on memristor-based computing [39], we simplify the interconnect line into a resistance-only device. In this way, the crossbar circuit is modified into a resistor network containing MN memristor cells (R_{act}), $2MN$ interconnect lines (r), and N sensing resistors (R_s).

C. Using the Average and Worst Cases for Estimation

When evaluating and optimizing a design, we need to know the overall accuracy loss instead of the results of a specific weight matrix or input sample. Therefore, MNSIM provides a coarse-grained accuracy model that analyzes the accuracy loss using the average and worse cases. For a crossbar with M rows and N columns, when the input voltages equal, the output voltage of a column is:

$$V_{\text{o}} = V_{\text{i}} \times \frac{R_{\text{s}}}{R_{\text{parallel}} + R_{\text{s}}} \tag{9}$$

where R_{parallel} is the parallel resistance of the whole column, and R_s is the equivalent sensing resistance. If we take the resistance of interconnect line between neighboring cells r into account, the R_{parallel} is larger than the parallel resistance of memristor cells, as shown in Equ. (10). Since r is much less than the resistance of memristor, the difference between denominators can be ignored. In the worst case, all memristors are at the minimum resistance, and the worst column is the farthest column from input signals. Therefore, the R_{parallel} can be approximately calculated by:

$$\frac{1}{R_{\text{parallel}}} \approx \sum_{m=1}^{M} \frac{1}{R_{m,n} + mr + nr} \approx \frac{M}{R_{\min} + (M+N)r} \tag{10}$$

where $R_{m,n}$ is the resistance of memristor cell in the mth row and the nth column, and $R_{\min}$ is the minimum resistance of memristor device. Take the non-linear V-I characteristics back into account, the practical resistance of memristor cell R_{act} differs from the ideal value R_{idl}. By substituting Equ. (10) into Equ. (9), we can estimate the actual output voltage. After simplification, the error of output voltage is:

$$V_{\text{o,idl}} - V_{\text{o,act}} = V_{\text{i}} \times \frac{[(M+N)r + R_{\text{act}} - R_{\text{idl}}]R_{\text{s}}M}{[R_{\text{act}} + (M+N)r + R_{\text{s}}M](R_{\text{idl}} + R_{\text{s}}M)} \tag{11}$$

The error rate is denoted by $(V_{\text{o,idl}} - V_{\text{o,act}})/V_{\text{o,idl}}$, which can be calculated using Equ. 11. We use M, N, and r as variables to simulate the error of output voltages on SPICE, and fit the relationship according to Equ. (11) to obtain the accuracy module, as shown in Fig. 5. The root mean squared error of this fitting curve is less than 0.01.

We can transform the above voltage error rate into the digital data deviation. The results of matrix-vector multiplication are linearly quantized into k levels by the read circuits. Therefore, given the quantization interval V_{interval}, the $k-1$ quantization boundaries are $\{0.5V_{\text{interval}}, ..., (k-1-0.5)V_{\text{interval}}\}$. According to the non-ideal factor analysis, the input data of these convert circuits contains a maximum deviation rate of ϵ, which causes a read deviation when converting an analog voltage into digital field. MNSIM uses both the worst-case deviation and the average deviation to evaluate the accuracy. In the worst case, the ideal computing result signal is around the largest quantization boundary $(k-1-0.5)V_{\text{interval}}$ and needs to be recognized as the maximum value $k-1$. The actual computing result in the worst case is $(k-1-0.5)V_{\text{interval}} \times (1-\epsilon)$, so the maximum deviation is $[(k-1-0.5)\epsilon + 0.5]V_{\text{interval}}$. The maximum digital deviation can be calculated by:

$$MaxDigitalDeviation = \lfloor (k-1.5)\epsilon + 0.5 \rfloor \tag{12}$$

For example, when k equals 64 and ϵ equals 10%, the $MaxDigitalDeviation$ equals 6, which means that the maximum value 63 can be wrongly read as 57. Therefore, the maximum error rate is:

$$MaxErrorRate = \frac{\lfloor (k-1.5)\epsilon + 0.5 \rfloor}{k-1} \tag{13}$$

For average case, the digital deviation of a specific quantization level i can be represented by $\lfloor i\epsilon + 0.5 \rfloor$, where we use i instead of $i-0.5$ or $i+0.5$ to reflect the average situation. Therefore, the average deviation is:

$$AvgDigitalDeviation = \frac{\sum_{i=0}^{k-1} \lfloor i\epsilon + 0.5 \rfloor}{k} \tag{14}$$

When simulating the application with multiple network layers, the input signal from the previous layer also has a fluctuation. Suppose that the digital error rate of the previous layer is δ_{d1} and the computing error rate of the current layer's crossbar is ϵ_{c2}, the practical analog output voltage of the current layer is limited by:

$$(1-\delta_{d1})(1-\epsilon_{c2})V_{\text{idl}} \le V_{\text{act}} \le (1+\delta_{d1})(1+\epsilon_{c2})V_{\text{idl}} \tag{15}$$

which can be substituted into Equ. (12)˜(14) to further analyze the read error rate of the current layer. MNSIM uses this propagation model to evaluate the final accuracy of the whole accelerator layer by layer.

TABLE II: Validation Results with Respect to a RRAM-Crossbar-Based Two Layer NN. The Technology Node of CMOS is 90nm.

Metric	MNSIM	SPICE Result	ERROR
Computation Power(mW) (Decoder+Crossbar)	17.20	16.34	+5.26%
Read Power(mW) (Decoder+Crossbar)	2.39	2.44	-2.05%
Computation Energy(uJ) (3-layer ANN)	0.525	0.487	+7.73%
Latency(ns)	381.49	405.50	-5.92%
Average Relative Accuracy(%)	95.41	94.57	**-0.89**%

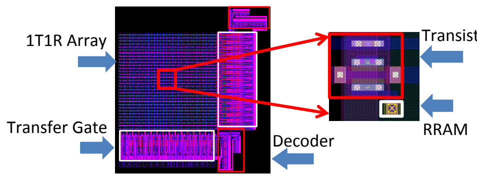


Fig. 6: The layout of 32×32 1T1R RRAM crossbar with the decoder in 130nm technology.

D. Device Variation

The device variation of a memristor cell has been modeled by previous research results [40]–[42]. A common used model is introducing a random factor into R_{act}. Given that σ is the maximum percentage of the random resistance deviation ranging from 0% to 30% in various devices [41], the worst case of practical resistance of memristor cell is $(1 \pm \sigma)R_{\text{act}}$. The deviation is an additional noise on resistance, which can be directly introduced into Equ. (9). After formula simplification similar to the variation-free case described in Section VI.C, we can derive the estimation model with resistance deviation σ, as:

$$\Delta V = \frac{V_{\text{i}}[(M+N)r + (1 \pm \sigma)R_{\text{act}} - R_{\text{idl}}]R_{\text{s}}M}{[(1 \pm \sigma)R_{\text{act}} + (M+N)r + R_{\text{s}}M](R_{\text{idl}} + R_{\text{s}}M)} \tag{16}$$

where $\Delta V = V_{\text{o,idl}} - V_{\text{o,act}}$. The verification result of the variation-considered model is similar to that shown in Fig. 5, because the method to introduce noise in SPICE is the same as that in MNSIM. The reference value of σ in MNSIM is set to 0 to provide the accuracy loss without noise, but the value can be modified by changing the $Memristor\_Model$ configuration to simulate the performance with variation.

VII. Experimental Results

A. Validation

The existed demonstrations of memristor-based computing [43], [44] are not suitable for validating the simulation models

TABLE III: Simulation Time of SPICE and MNSIM.

Crossbar Size	16	32	64	128	256
SPICE(s)	5.35	13.76	41.62	169.12	678.2
MNSIM(s)	0.0007	0.0011	0.0030	0.0192	0.0348
Speed-Up	7642×	12509×	13873×	8088×	19489×

TABLE IV: The Design Space Exploration of the Large Computation Bank Case (a 2048×1024 Network Layer).

Large Computation Bank Case	**Area**	**Energy**	**Latency**	**Computation Accuracy**
Area(mm^2)	**12.18**	20.73	29.35	117.1
Energy per Input Sample(uJ)	35.90	**3.192**	3.748	10.35
Latency(us)	43.43	0.5153	**0.3470**	10.35
Error Rate of Output(%)	17.98	17.98	17.98	**1.090**
Power(W)	0.8266	6.195	10.80	29.66
Crossbar Size	256	256	256	64
Line Tech Node	28	28	28	45
Parallelism Degree	1	128	256	64

because: 1) the detailed power and latency are not published; 2) these chips only contain the computational memristor crossbars, and the peripheral functions are processed by off-chip CPUs or FPGAs. Therefore, we use the simulation results from SPICE to validate the models in MNSIM. We choose a 3-layer fully-connected NN with two 128×128 network layers as the application to validate the power and latency module. The SPICE results are the average value of 20 random samples of weight matrices, where 100 random input samples are generated for each network. The technology node of CMOS is 90nm. The results are shown in Table II. All the error rates are smaller than 10% compared with the SPICE results. The simulation error on power consumption is caused by the average-case model of input voltages and memristor cell resistances in MNSIM, which further influences the simulation result on energy consumption. As for the validation of accuracy loss part, we use an approximate computing application, the JPEG encoding processed in a 3-layer $64\times16\times64$ neural network [12]. The result shows the error rate of accuracy model is less than 1%.

Since only the crossbar and the decoder are designed by MNSIM to support neuromorphic computation, we use the parameter extracted from the layout to validate the area model of this part. As shown in Fig. 6, a 32×32 1T1R memristor crossbar with the computation-oriented decoders are designed in 130nm CMOS technology. The area of the layout is 3420 um^2 (45um × 76um), while the estimation result is 2251 um^2. The large error rate of area estimation is mainly caused by the reason that the layout design needs to remain some extra space. MNSIM introduces the validation result as a coefficient for area estimation. Users can provide the coefficient of their own technologies to obtain a more accurate estimation.

B. Simulation Speed-up

We test the simulation time of single memristor crossbar with different sizes. As shown in Table III, MNSIM obtains more than $7000\times$ speed-up compared with SPICE. The speed-up will further increase when simulating accelerator with multiple crossbars and a large number of peripheral circuits.

TABLE V: The Trade-Off between Area, Power, and Accuracy Based on Different Crossbar Sizes.

Crossbar Size	256	128	**64**	32	16	8
Error Rate(%)	7.71	2.07	**1.09**	1.46	2.38	3.50
Area(mm^2)	29.34	58.59	117.11	234.10	468.32	936.81
Energy(uJ)	3.74	5.94	10.35	19.21	37.09	73.38

C. Case Study: Large Computation Bank

We use a 2048×1024 fully-connected layer to evaluate the optimization of a large neural network layer. The data precision is set to 4-bit signed weights and 8-bit signals [14], [24], [28]. This case study is based on 7-bit level memristor model [45], which supports at most 8-bit signed weights in two memristor crossbars.

We use the reference design based on 45nm CMOS. The crossbar size, computation parallelism degree, and interconnect technology are three variables for design space exploration. The computation parallelism degree means the number of read circuits for each crossbar. For example, when the parallelism degree is 4, it means we simultaneously obtain the results of 4 columns for each crossbar. In this experiment, the crossbar size doubled increases from 4, 8, to 1024; the computation parallelism degree ranges from 1 to 128; and the interconnect technology (nm) is chosen from $\{18,22,28,36,45\}$. MNSIM uses traversal method for optimization taking advantage of the high simulation speed. All the 10,220 designs are simulated within 4 seconds in this case.

1) Design Space Exploration: We set a constraint that the computing error rate of memristor crossbar cannot be larger than 25% in the experiment. The design space exploration results are shown in Table V. Each column of the table shows the performance and the design parameters of an optimal design aimed at a specific optimization target. Compared with the 2nd and 3rd columns, the 1st column has less area and power consumption with the same interconnect technology and crossbar size. This is because it reads the computing results one by one, but the latency increases and the energy of entire computation grows back. From the 4th column, we find that the most accurate computation is implemented by large interconnect technology and a middle crossbar size, which matches our previous analysis. Since changing digital modules does not impact the computing accuracy of memristor crossbars, the user can set a secondary optimization target for accuracy optimization. Fig. 9(a) compares the four optimal design in Table V. Each pentagon shows the reciprocal area, energy efficiency, reciprocal power, speed, and accuracy of an optimal design. The reciprocal area, energy efficiency, reciprocal power, and speed factors are normalized by the maximum value. The results show that the values of performance factors vary a lot in different optimization targets, and directly optimizing a single performance factor usually means that the value of other factors will be low. Therefore, MNSIM also supports the trade-off analysis to obtain a compromised result among all performance factors.

2) Trade-Off Among Area, Power, and Accuracy: Since each splitting of rows in a weight matrix leads to additional peripheral circuits like ADCs, the power and area decrease

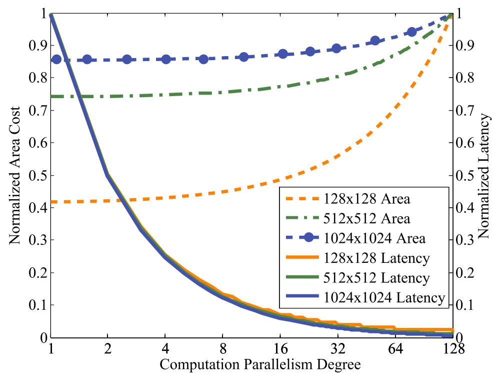


Fig. 7: The influence of computing parallelism degree on area and latency with different crossbar sizes. The area and latency results are normalized by the maximum value of each crossbar size for comparison.

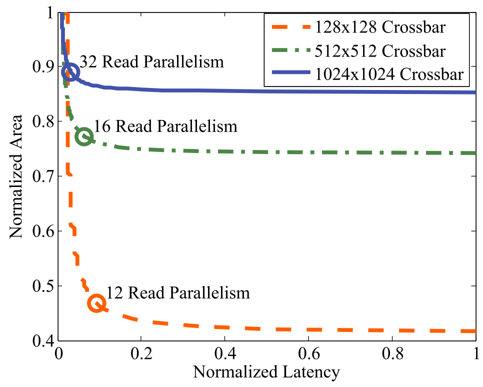


Fig. 8: The trade-off relationship between area and latency with different computation parallelism degrees and different crossbar sizes.

when using larger crossbar. However, large crossbar suffers from the impact of non-ideal factors, so there is a trade-off between computing accuracy and other performances. We use the 45nm interconnect technology as an example to analyze the trade-off relationship. Table IV shows that we can get accuracy improvement at the cost of area and power only when the crossbar size is larger than 64. When the crossbar is too small, the parallel resistance of a column grows up, which leads to the resistance deviation by the non-linear V-I characteristic of memristor [39].

3) Trade-Off Between Latency and Area: There is also a trade-off between latency and area. The area and power can be reduced by sharing the output peripheral circuits but the latency increases. Fig. 7 shows the area and latency results when using different computation parallelism degrees and different crossbar sizes, where each line shows the results of the same crossbar size. We normalize the results by the maximum value of each crossbar size's result. When the parallelism degree goes down, the increasing trend of latency is similar in different crossbar sizes, but the area reduction trend varies. This is because the number of computation units is few when using large crossbar size. Therefore, the area of neurons and peripheral circuits takes a large proportion of area, which limits the gain of reducing read circuits. The trade-off between area and latency is shown in Fig. 8. We can obtain large area reduction at the cost of little latency, and there is an inflection point for each crossbar size.

TABLE VI: The Design Space Exploration of the CNN Case (VGG-16 [46] Network).

CNN Case	Area	Energy	Latency	Computation Accuracy
Area(mm^2)	**164.9**	390.7	553.1	328.8
Energy per Sample(mJ)	349.5	**9.718**	11.41	230.6
Latency per Pipeline Cycle (us)	21.81	0.5217	**0.3513**	10.9968
Error Rate of Output(%)	43.92	43.92	43.92	**12.49**
Power(W)	72.98	170.2	296.7	155.1
Crossbar Size	128	128	128	64
Line Tech Node	45	45	45	90
Parallelism Degree	1	128	256	64

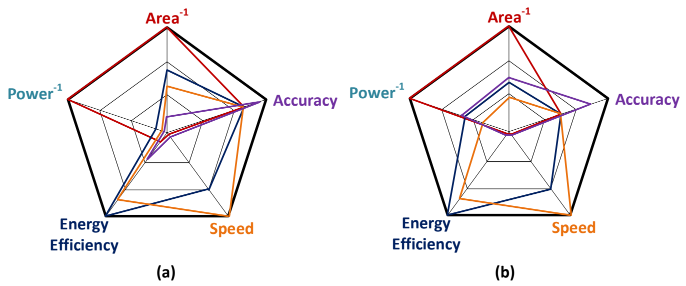


Fig. 9: The performances of the optimal designs for (a) large computation bank and (b) deep CNN.

D. Case Study: Deep CNN

We use VGG-16 network [46] on ImageNet [47] as a CNN case study. The precision values of weights and intermediate data are set at 8-bit according to current quantization result [14]. The precision of memristor device is set at 7-bit memristor model [48], and the CMOS technology node is 45nm. In this case study, the interconnect line technology, computation parallelism degree, and the crossbar size are set as common variables in the entire accelerator level.

We reduce the error rate constraint to 50% and enlarge the interconnect range up to 90nm. We set the ranges of other parameters for optimization at the same ranges as used in Section VII.C. The whole design space exploration results are shown in Table VI. The latency is defined as the latency of each pipeline cycle, which is determined by the latency of the largest Computation Bank. The optimal parallelism degree results for an entire CNN is the same as that for a computation Bank due to the pipeline structure. The optimal crossbar sizes and interconnect line technique nodes are different when considering the accumulation of errors. Fig. 9(b) shows the four optimal design. Compared with the single neural layer, the entire network case has a smaller difference in different optimal designs.

E. Case Study: Simulation for Related Work

Two related designs [6], [7] are simulated using MNSIM to validate its scalability. The results of two work are not comparable because the scale and the detailed structures are much different.

TABLE VII: The Simulation of PRIME [6] and ISAAC [7]. The results of the two work are not comparable because the neural network scales used in the two cases are much different.

Work	**PRIME [6]**	**ISAAC [7]**
CMOS Tech	65nm	32nm
Structure	FF-subarray	ISAAC Tile
Area (mm^2)	0.17	0.37
Energy per Task (uJ)	0.08	0.94
Latency (us)	0.66	2.2
Accuracy	91%	96%

1) PRIME: For PRIME, all the modules in the FF subarray have been modeled in MNSIM, but the connection of the modules is not directly supported by the reference design. We customized the connections by moving the adders, neurons, pooling buffers, and pooling functions into the computation units to simulate the reconfigurable units in PRIME. We use the same simulation configurations as the experiments in the PRIME [6]. The device is RRAM, and the crossbar size is 256. The input and output data are 6-bit fixed point, and the precision of ADC is also 6-bit. The weights are 8-bit signed values, and each RRAM cell can store 4-bit unsigned weight, which means four memristor cells are needed to store a weight. The technology node of CMOS is 65nm.

PRIME is an entire architectural work, whose simulation containing CPU, Memory, and the data transportation cost on the bus. But MNSIM only focuses on the simulation of accelerator part, namely the FF-subarrays [6]. Therefore, we simulate the performance of an FF-subarray for this case study. The results are shown in Table VII. A computation task with a 256$\times$256 DNN layers is used to evaluate the peak performance of an FF-subarray with four crossbars.

2) ISAAC: In ISAAC, some modules are not contained in the reference design of MNSIM, such as the S&H module, the eDRAM buffer, and the customized DAC/ADC module. The authors have provided the dynamic power and area consumption of these modules, and we directly import them into the simulation. For latency and leakage power simulation, we use the circuits in [49], [50] as the reference design and import the performance model into the simulation. The other configurations are the same as the experiments in ISAAC. The technology node of CMOS is 32nm. The memristor crossbar size is 128. Since the authors haven't provided the detailed device information of the memristor cells, we use RRAM in this case study.

We simulate the performance of an ISAAC Tile in [7] for this case study. A special design of ISAAC is the 22-level inner pipeline structure in an ISAAC Tile, which is much different from the entire-parallel scheme of the reference design in MNSIM. The area can be directly simulated by the current simulation flow, but the latency simulation needs to be customized to find the critical path of the inner pipeline. After that, the energy consumption of the 22 cycles can be simulated using the power of each module. The results are shown in Table VII. The area result is the same as the value in the original publication. This is because the majority of the area is consumed by DACs, ADCs, and eDRAMs, and these modules are regarded as customized modules whose area consumption are introduced from the original publication. A large-enough computation task that uses all the 96 crossbars in an ISAAC Tile is chosen to evaluate the other performances. The energy consumption and latency are the accumulated values for the 22 pipelined cycles, which accord with the values in the original publication [7].

VIII. Conclusions

In this work, we have presented the first behavior-level simulation platform for the memristor-based neuromorphic computing system. MNSIM proposes a hierarchical structure of memristor-based neuromorphic computing accelerator, and provides flexible interfaces to customize the design in multiple levels. A behavior-level model is proposed to estimate the computing accuracy of the memristor-based structure. The experiment results show that MNSIM reaches more than 7000$\times$ speed-up compared with SPICE. MNSIM also provides the trade-off between different designs and estimates the optimal performance.

In the future, we will further support the simulation for other structures like dynamic synaptic properties [22], on-chip Training method [51], and inner-layer pipeline structure [7].

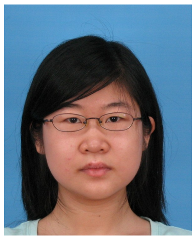
References

[1] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, pages 436–444, 2015.
[2] Hadi Esmaeilzadeh et al. Neural acceleration for general-purpose approximate programs. In *International Symposium on Microarchitecture*, pages 449–460, 2012.
[3] M Mitchell Waldrop. The chips are down for moores law. *Nature News*, page 144, 2016.
[4] Yuan Xie. Future memory and interconnect technologies. In *DATE*, pages 964–969, 2013.
[5] Sung Hyun Jo et al. Nanoscale memristor device as synapse in neuromorphic systems. *Nano letters*, pages 1297–1301, 2010.
[6] Ping Chi et al. PRIME: A novel processing-in-memory architecture for neural network computation in ReRAM-based main memory. In *ISCA*, 2016.
[7] Ali Shafiee et al. ISAAC: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars. In *ISCA*, 2016.
[8] Nathan Binkert et al. The gem5 simulator. *ACM SIGARCH Computer Architecture News*, pages 1–7, 2011.
[9] Xiangyu Dong et al. NVSim: A circuit-level performance, energy, and area model for emerging non-volatile memory. In *Emerging Memory Technologies*, pages 15–50. 2014.
[10] Matt Poremba and Yuan Xie. NVMain: An architectural-level main memory simulator for emerging non-volatile memories. In *IEEE Computer Society Annual Symposium on VLSI*, pages 392–397, 2012.
[11] Wei Fei et al. Design exploration of hybrid CMOS and memristor circuit by new modified nodal analysis. *IEEE Trans. VLSI*, pages 1012–1025, 2012.
[12] Boxun Li et al. RRAM-based analog approximate computing. *IEEE Trans. CAD*, pages 1905–1917, 2015.
[13] Christian Szegedy et al. Going deeper with convolutions. In *CVPR*, pages 1–9, 2015.
[14] Jiantao Qiu et al. Going deeper with embedded fpga platform for convolutional neural network. In *FPGA*, pages 26–35, 2016.
[15] Brahmantyo Heruseto et al. Embedded analog CMOS neural network inside high speed camera. In *ASQED*, pages 144–147, 2009.
[16] Jue Wang et al. i2WAP: Improving non-volatile cache lifetime by reducing inter-and intra-set write variations. In *HPCA*, pages 234–245, 2013.
[17] Miao Hu et al. Hardware realization of BSB recall function using memristor crossbar arrays. In *DAC*, pages 498–503, 2012.
[18] Miao Hu et al. Memristor crossbar-based neuromorphic computing system: A case study. *IEEE Transactions on neural networks and learning systems*, pages 1864–1878, 2014.
[19] Xiaoxiao Liu et al. A heterogeneous computing system with memristor-based neuromorphic accelerators. In *HPEC*, pages 1–6, 2014.

[20] Wolfgang Maass. Networks of spiking neurons: the third generation of neural network models. *Neural networks*, pages 1659–1671, 1997.
[21] Tianqi Tang et al. Spiking neural network with RRAM: Can we use it for real-world application? In *DATE*, pages 860–865, 2015.
[22] Miao Hu et al. A memristor-based dynamic synapse for spiking neural networks. *IEEE Trans. CAD*, 2016.
[23] Yu Wang et al. Low power convolutional neural networks on a chip. In *ISCAS*, pages 129–132, 2016.
[24] Lixue Xia et al. Switched by input: power efficient structure for RRAM-based convolutional neural network. In *DAC*, page 125, 2016.
[25] Ligang Gao, Pai-Yu Chen, and Shimeng Yu. Demonstration of convolution kernel operation on resistive cross-point array. *IEEE Electron Device Letters*, pages 870–873, 2016.
[26] Meng-Fan Chang et al. A low-power subthreshold-to-superthreshold level-shifter for sub-0.5V embedded resistive RAM (ReRAM) macro in ultra low-voltage chips. In *APCCAS*, pages 695–698, 2014.
[27] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.
[28] Binary convolutional neural network on RRAM, author=Tang, Tianqi and others, booktitle=ASP-DAC, pages=782–787, year=2017.
[29] Shaoli Liu et al. Cambricon: An instruction set architecture for neural networks. In *ISCA*, pages 393–405, 2016.
[30] Boxun Li et al. Merging the interface: Power, area and accuracy co-optimization for rram crossbar-based mixed-signal computing system. In *DAC*, page 13. ACM, 2015.
[31] Makoto Itoh and Leon O Chua. Memristor oscillators. *International Journal of Bifurcation and Chaos*, pages 3183–3206, 2008.
[32] Y Sasago et al. Cross-point phase change memory with 4F2 cell size driven by low-contact-resistivity poly-si diode. In *VLSI Symp.*, pages 24–25, 2009.
[33] A 4Mb embedded SLC resistive-RAM macro with 7.2 ns read-write random-access time and 160ns MLC-access capability, author=Sheu, Shyh-Shyuan and others, booktitle=ISSCC, pages=200–202, year=2011.
[34] Jing Li et al. A novel reconfigurable sensing scheme for variable level storage in phase change memory. In *IMW*, pages 1–4, 2011.
[35] Boris Murmann. ADC performance survey 1997-2016, 2015.
[36] S. J. E. Wilton and N. P. Jouppi. CACTI: an enhanced cache access and cycle time model. *IEEE JSSC*, pages 677–688, 1996.
[37] Wei Zhao and Yu Cao. Predictive technology model for nano-cmos design exploration. *ACM JETC*, page 1, 2007.
[38] Saurabh Sinha et al. Exploring sub-20nm finfet design with predictive technology models. In *DAC*, pages 283–288, 2012.
[39] Lixue Xia et al. Technological exploration of RRAM crossbar array for matrix-vector multiplication. *Journal of Computer Science and Technology*, pages 3–19, 2016.
[40] Miao Hu et al. Dot-product engine for neuromorphic computing: programming 1T1M crossbar to accelerate matrix-vector multiplication. In *DAC*, pages 1–6, 2016.
[41] Beiye Liu et al. Digital-assisted noise-eliminating training for memristor crossbar-based analog neuromorphic computing engine. In *DAC*, pages 1–6, 2013.
[42] Shinhyun Choi, Yuchao Yang, and Wei Lu. Random telegraph noise and resistance switching analysis of oxide based resistive memory. *Nanoscale*, pages 400–404, 2014.
[43] Daeseok Lee et al. Oxide based nanoscale analog synapse device for neural signal recognition system. In *IEDM*, pages 4–7, 2015.
[44] Myonglae Chu et al. Neuromorphic hardware system for visual pattern recognition with memristor array and CMOS neuron. *IEEE Transactions on Industrial Electronics*, pages 2410–2419, 2015.
[45] Ligang Gao, F Alibart, and DB Strukov. A high resolution nonvolatile analog memory ionic devices. In *NVMW*, pages paper–57, 2013.
[46] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.
[47] Jia Deng et al. Imagenet: A large-scale hierarchical image database. In *CVPR*, pages 248–255, 2009.
[48] Fabien Alibart et al. High precision tuning of state for memristive devices by adaptable variation-tolerant algorithm. *Nanotechnology*, page 075201, 2012.
[49] Micah O'Halloran and Rahul Sarpeshkar. A 10-nW 12-bit accurate analog storage cell with 10-aA leakage. *IEEE JSSC*, pages 1985–1996, 2004.
[50] Lukas Kull et al. A 3.1 mW 8b 1.2 GS/s single-channel asynchronous SAR ADC with alternate comparators for enhanced speed in 32 nm digital soi cmos. *IEEE JSSC*, pages 3049–3058, 2013.
[51] Mirko Prezioso et al. Training and operation of an integrated neuromorphic network based on metal-oxide memristors. *Nature*, pages 61–64, 2015.

Lixue Xia (S'14) received the B.S. degree in Electronic Engineering from Tsinghua University, Beijing, in 2013. He is currently pursuing his Ph.D. degree in Department of Electronic Engineering, Tsinghua University. His research mainly focuses on energy efficient hardware computing system design, and neuromorphic computing system based on emerging Non-volatile devices.

Boxun Li (S'13) received the B.S. degree and the M.S. degree in Electronic Engineering from Tsinghua University in 2013, and in 2016. His research mainly focuses on energy efficient hardware computing system design, and parallel computing based on GPU.



Tianqi Tang received the B.S. degree in Electronic Engineering from Tsinghua University, Beijing, in 2014. She is currently pursuing her M.S. degree in Department of Electronic Engineering, Tsinghua University. Her research mainly focuses on On-Chip Neural Network System and Emerging Non-Volatile-Memory Technology.

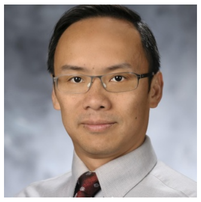
Peng Gu (S'15) received the B.S. degree in Electronic Engineering from Tsinghua University, Beijing, in 2015. He is currently pursuing the Ph.D. degree with the SEALab, Department of Electrical and Computer Engineering, University of California, Santa Barbara, CA, USA. His current research interests include low power system design and hardware acceleration and computing with emerging devices. He has authored and co-authored several papers in DAC, DATE and ASP-DAC.

Pai-yu Chen (S'15) received the B.S. degree and the M.S.E. degree in electrical engineering from National Taiwan University in 2010 and University of Texas at Austin in 2013. He is currently working towards the Ph.D. degree in electrical engineering at Arizona State University. His research interests include emerging non-volatile memory device/architecture design, new computing paradigm exploration, and hardware design for security system.

Shimeng Yu (S'10-M'14) received the B.S. degree in microelectronics from Peking University in 2009, and the M.S. degree and Ph.D. degree in electrical engineering from Stanford University in 2011, and in 2013, respectively. He is currently an assistant professor of electrical engineering and computer engineering at Arizona State University.
His research interests are emerging nano-devices and circuits with focus on the resistive switching memories, and new computing paradigms with focus on the neuro-inspired computing. Among this honors, he was awarded the IEEE Electron Devices Society PhD Student Fellowship in 2012, the DoD-DTRA Young Investigator Award in 2015, the NSF Faculty Early CAREER Award in 2016, and the ASU Fulton Outstanding Assistant Professor in 2017.

Yu Cao (S'99-M'02-SM'09-F'17) received the B.S. degree in physics from Peking University in 1996. He received the M.A. degree in biophysics and the Ph.D. degree in electrical engineering from University of California, Berkeley, in 1999 and 2002, respectively.
He worked as a summer intern at Hewlett-Packard Labs, Palo Alto, CA in 2000, and at IBM Microelectronics Division, East Fishkill, NY, in 2001. After working as a post-doctoral researcher at the Berkeley Wireless Research Center (BWRC), he is now a Professor of Electrical Engineering at Arizona State University, Tempe, Arizona. He has published numerous articles and two books on nano-CMOS modeling and physical design. His research interests include physical modeling of nanoscale technologies, design solutions for variability and reliability, reliable integration of post-silicon technologies, and hardware design for on-chip learning.
Dr. Cao was a recipient of the 2012 Best Paper Award at IEEE Computer Society Annual Symposium on VLSI, the 2010, 2012, 2013, 2015 and 2016 Top 5% Teaching Award, Schools of Engineering, Arizona State University, 2009 ACM SIGDA Outstanding New Faculty Award, 2009 Promotion and Tenure Faculty Exemplar, Arizona State University, 2009 Distinguished Lecturer of IEEE Circuits and Systems Society, 2008 Chunhui Award for outstanding oversea Chinese scholars, the 2007 Best Paper Award at International Symposium on Low Power Electronics and Design, the 2006 NSF CAREER Award, the 2006 and 2007 IBM Faculty Award, the 2004 Best Paper Award at International Symposium on Quality Electronic Design, and the 2000 Beatrice Winner Award at International Solid-State Circuits Conference. He has served as Associate Editor of the IEEE Transactions on CAD, and on the technical program committee of many conferences.

Yu Wang (S05-M07-SM14) received his B.S. degree in 2002 and Ph.D. degree (with honor) in 2007 from Tsinghua University, Beijing. He is currently a tenured Associate Professor with the Department of Electronic Engineering, Tsinghua University. His research interests include brain inspired computing, application specific hardware computing, parallel circuit analysis, and power/reliability aware system design methodology.
Dr.Wang has authored and coauthored over 150 papers in refereed journals and conferences. He has received Best Paper Award in FPGA 2017, ISVLSI 2012, and Best Poster Award in HEART 2012 with 7 Best Paper Nominations (ASPDAC 2014, ASPDAC 2012, 2 in ASPDAC 2010, ISLPED 2009, CODES 2009). He is a recipient of IBM X10 Faculty Award in 2010. He served as TPC chair for ICFPT 2011 and Finance Chair of ISLPED 2012-2016, and served as program committee member for leading conferences in these areas, including top EDA conferences such as DAC, DATE, ICCAD, ASP-DAC, and top FPGA conferences such as FPGA and FPT.
Currently he serves as Co-Editor-in-Chief for ACM SIGDA E-Newsletter, Associate Editor for IEEE Transactions on CAD, and Journal of Circuits, Systems, and Computers. He also serves as guest editor for Integration, the VLSI Journal and IEEE Transactions on Multi-Scale Computing Systems. He has given 25+ invited talks in industry/academia. He is now with ACM Distinguished Speaker Program. He is an ACM/IEEE Senior Member.



Yuan Xie (F15) was with IBM, Armonk, NY, USA, from 2002 to 2003, and AMD Research China Lab, Beijing, China, from 2012 to 2013. He has been a Professor with Pennsylvania State University, State College, PA, USA, since 2003. He is currently a Professor with the Electrical and Computer Engineering Department, University of California at Santa Barbara, Santa Barbara, CA, USA. His current research interests include computer architecture, Electronic Design Automation, and VLSI design.

Huazhong Yang (M'97-SM'00) was born in Ziyang, Sichuan Province, P.R.China, on Aug.18, 1967. He received B.S. degree in microelectronics in 1989, M.S. and Ph.D. degree in electronic engineering in 1993 and 1998, respectively, all from Tsinghua University, Beijing.
In 1993, he joined the Department of Electronic Engineering, Tsinghua University, Beijing, where he has been a Full Professor since 1998. Dr. Yang was awarded the Distinguished Young Researcher by NSFC in 2000 and Cheung Kong Scholar by Ministry of Education of China in 2012. He has been in charge of several projects, including projects sponsored by the national science and technology major project, 863 program, NSFC, 9th five-year national program and several international research projects. Dr. Yang has authored and co-authored over 400 technical papers, 7 books, and over 100 granted Chinese patents. His current research interests include wireless sensor networks, data converters, energy-harvesting circuits, nonvolatile processors, and brain inspired computing. Dr. Yang has also served as the chair of Northern China ACM SIGDA Chapter.