Input-Aware Flow-Based Computing on Memristor Crossbars With Applications to Edge Detection

Dwaipayan Chakraborty, *Student Member, IEEE*, Sunny Raj, *Student Member, IEEE*, Steven Lawrence Fernandes, *Member, IEEE*, and Sumit Kumar Jha, *Member, IEEE*

***Abstract*— Sneak paths in nanoscale memristor crossbars have traditionally been viewed as a problem in the use of memristor crossbars as non-volatile replacements of traditional volatile RAM memories. We show that the sneak paths in a memristor crossbar can be employed to perform computation that exploits device-level parallelism. Our computation can be performed in the memory and does not require data to be moved between a processor and a memory unit – thereby, avoiding the von Neumann bottleneck. We demonstrate the potential of our approach by applying it to a basic problem in computer vision: edge detection in an image. Our results show that the flow-based computing approach on nanoscale memristor crossbars can be used to obtain high-quality approximations of edge detection. We have synthesized multiple 8 × 8 crossbar circuits for this purpose – a single crossbar circuit for detecting edges between all possible pixel pairs with ∼85% accuracy, and another family of input-aware crossbars with higher performance over real-world images. The family of input-aware crossbars together performs approximate edge detection for a subset of pixel pairs obtained from analyzing the BSD500 database, and the resultant images are of a quality comparable to exact edge detection.**

***Index Terms*— Nanoscale, memristor, crossbar, flow-based computing, edge detection, AI, input-aware computing, approximate computing.**

I. Introduction

THE rise of digital data acquisition such as digital cameras has led to a deluge of high-volume high-velocity big data whose subsequent processing requires both the ability to process this high-velocity data quickly and the capability to rapidly transfer this high-volume data from the memory to the processor. In many intelligent applications such as smart cameras, the computation being performed is fixed *a priori* such as edge detection or face identification. Our proposed approach enables the design of nanoscale memristor crossbars for implementing such *a priori* known computations using naturally occurring sneak paths in two-dimensional arrays of nanoscale memristors.

Artificial Intelligence (AI) applications often rely on high-dimensional image data and there is an urgent need to eliminate or reduce the von Neumann bottleneck that requires this data to be moved from memory to the processor and back. Significant progress has been made towards reducing the von Neumann bottleneck by creating high-performance memories. The crossbar circuit topology is making its way into mainstream technologies [1]–[4], because of its design as a two-dimensional uniform array of devices and the subsequent ease of fabrication. Significant efforts have been made towards efficient fabrication of the crossbar circuits [5]–[10]. A combination of the above developments has led to the production of high-performance memory devices on the crossbar fabric [11]–[14]. While these advancements have definitely yielded performance and scalability improvements, the separation between the processor and the memory remains a bottleneck in both computing speed and energy efficiency.

Our flow-based computing approach stores data on a nanoscale memristor crossbar in such a pattern that the flow of current through the naturally-occurring sneak paths in the crossbar performs the desired Boolean computation. The nanoscale memristor crossbar is used both for the storage of the data and for performing the desired computation – thus, eliminating the von Neumann bottleneck.

Flow-based computing designs involving only a few memristors can be obtained by implicitly or explicitly enumerating the space of all possible designs [15]. However, flow-based computing memristor crossbar designs for even simple AI applications such as edge detection cannot be synthesized using such direct enumeration approaches. An overview of our proposed approach is illustrated in Figure 1. In this article, we make the following contributions:

- We demonstrate how flow-based computing on nanoscale memristor crossbars [16] coupled with a simulated annealing search based on model counting [17] can be used to design crossbars that approximately implement edge detection in images. Our approach [17] is the first to use model counting to search the space of possible designs.
- We show that the results of a number of approximately correct input-aware memristor crossbars can be combined using elementary logical operations implemented

Manuscript received May 1, 2019; revised July 4, 2019; accepted July 31, 2019. Date of publication August 7, 2019; date of current version September 17, 2019. This work was supported in part by the National Science Foundation under Award 1822976 and Award 1422257, in part by the Florida Cybersecurity Center, in part by the Royal Bank of Canada, in part by the Air Force Young Investigator Award, and in part by the University of Central Florida Preeminent Post-doctoral Fellowship Program. This article was presented in part at the 2017 IEEE International Symposium on Circuits and Systems (ISCAS) and in part at the 2017 Design, Automation & Test in Europe Conference & Exhibition (DATE). This paper was recommended by Guest Editor X. Zhang. *(Corresponding author: Sumit Kumar Jha.)*

The authors are with the Computer Science Department, University of Central Florida, Orlando, FL 32816 USA (e-mail: dchakra@eecs.ucf.edu; sraj@eecs.ucf.edu; steven@eecs.ucf.edu; jha@eecs.ucf.edu).

Color versions of one or more of the figures in this article are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/JETCAS.2019.2933774

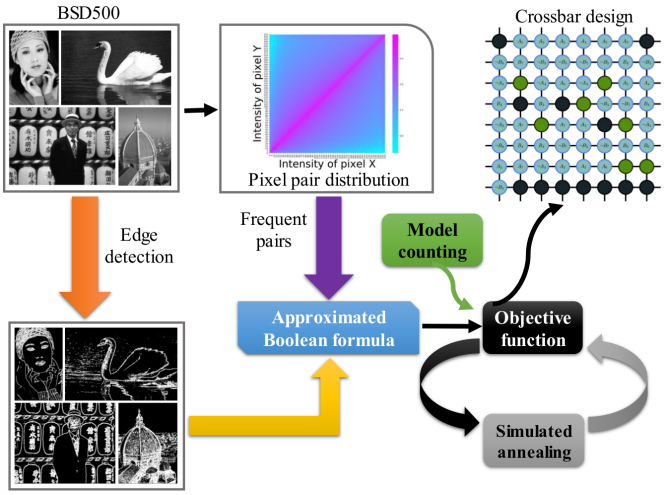


Fig. 1. Automated input-aware synthesis of memristor crossbars using model counting and simulated annealing. An approximately correct crossbar design is obtained by searching the space of candidate designs using simulated annealing. Each design is compared against the approximate Boolean formula learned from a real-world data set using model counting.

on memristor crossbars to enable high-quality edge detection in real-world images.

Our results indicate that the flow-based computing approach may be applicable to more interesting AI applications, such as deep learning. The rest of the paper is organized as follows: Section II discusses related work and presents it in the context of the current manuscript. Section III presents the idea of flow-based computing on nanoscale memristor crossbars. Our approach for searching the space of possible nanoscale memristor crossbar designs using model counting is presented in Section IV. Section V presents the results of applying flow-based computing to the problem of edge detection, and we conclude with ideas for future work in Section VI.

II. Related Work

The expected demise of Moore's law has fueled active research in the area of emerging computer architectures over the last decade. This problem has been further exaggerated by the rise of big data and the consequent narrowing of the von Neumann bottleneck between the processor and the memory. We briefly survey both these topics in Sections II-A and II-B.

Our proposed solution is based on the use of nanoscale memristors as non-volatile switches storing the data values and guiding the flows of current through the nanoscale memristor crossbar to implement a desired computation. Section II-C introduces memristors and their dynamics, and Section II-D briefly describes other efforts that have used nanoscale memristor crossbars for implementing in-memory computing. Our survey of related work is certainly neither exhaustive nor representative; it is only an attempt at providing a general perspective to the potential readers of this manuscript.

A. Moore's Law and Its Limitations

Over the last five decades, Moore's Law [18], [19] has been a driving factor towards immense technological and societal progress. Akin to how Moore's Law paints a picture for the future of chip design, Dennard scaling [20] examines possibilities for the scaling of devices. Dennard scaling hit a roadblock as CMOS transistors started replacing MOSFETs as this has been marked by a shift from micrometer (μm) scale to nanometer (nm) scale devices.

Interesting breakthroughs have been made in the area by using self-aligned double gate MOSFET structures (FinFET) [21], which are projected to scale down to about 10nm or smaller. The chip density and the energy efficiency of CMOS-based architectures have been increased by three dimensional layering of active devices [22] and by intuitively utilizing voltage scaling techniques [23], respectively. As the device dimensions are reduced, the traditional challenges posed by capacitance and resistance become more daunting, dielectric properties of materials cause more interference, and physical properties like orientation of silicon lattice structures, wafer thickness and mechanical strain start having more effects on the device behavior.

Our flow-based computing approach does not require the repeated switching of devices to perform computation and exploits the natural flow of current through devices storing data to produce results. By not relying on repeated device switching during a computation, our approach is less affected by Dennard scaling and dark silicon. We believe that this property makes our flow-based computing approach different from many other memristor-based approaches that require repeated switching of memristor devices [4], [24]–[26] for logical computations.

B. John Von Neumann Bottleneck and Beyond

John von Neumann's "First Draft of a Report on the EDVAC" [27] first circulated in 1945 defined a new computer architecture that has thrived for the last 74 years. The stored-program computer became the *de facto* computer and was widely replicated for commercial use. The architecture can be explained very simply by breaking it down into its modular components – primarily the central processing unit and the memory. The von Neumann computer can be interfaced with input and output devices. The mathematical and engineering foundations for this architecture were subsequently laid in [28].

A weakness in the von Neumann architecture is the processor-memory bottleneck, as discussed in [29], [30]. The bus connecting the processor and the memory will always have a limited bandwidth, thereby throttling performance. Despite the dynamic random access memory (DRAM) performance improving exponentially, it has been shown [31] that the processor-memory gap will also increase in an exponential manner – ultimately hitting a memory wall.

An inelegant but useful solution to the von Neumann bottleneck was the use of on-chip cache memory as a component of the existing memory hierarchy [32], which has since become an industry standard for designing microprocessors. Alternative approaches towards the nullification of the von Neumann bottleneck include the data-flow based programming paradigm [33], the processor-in-memory (PIM) architecture [34], in-memory computation of logic operations [35], and the end-to-end computation-in-memory (CIM) model [36].

Our approach completely nullifies the von Neumann bottleneck by storing data in a specific pattern on the memristor

crossbar and using only the flow of current through naturally occurring sneak paths in the crossbar to perform the desired computation. Our solution still uses von Neumann's stored program concept and relies on a classical von Neumann computer to compile a program into the desired pattern once such that the data should be loaded onto the crossbar using such a pattern to implement the desired program [37].

C. Memristor – the Fourth Fundamental Electrical Element

In 1971, Leon Chua used symmetry arguments to hypothesize that there exists a fourth fundamental electrical element in addition to the known elements: resistors, capacitors and inductors. This new element, which demonstrated a relation between flux linkage and electrical charge, was termed as a "memristor" [38] – an amalgamation of memory and resistor. This name arose from the fact that a memristor's resistance varies with the net current flowing through it and the resistance state can be maintained even if the device is disconnected from a power source, thereby giving it the property of non-volatile memory. Given the voltage v, the current i, the charge q and the flux linkage ϕ, one can define $d\phi(q)$ as the change in flux linkage. The memristance $M(q)$ of a memristor is defined as follows:

$$M(q) = \frac{d\phi(q)}{dq} \tag{1}$$

The voltage across a memristor at time t is characterized as:

$$v(t) = M(q(t))i(t) \tag{2}$$

The voltage drop across the memristor decreases and the current across it increases as it transitions from a high resistance state (HRS) to a low resistance state (LRS), when an external positive voltage is applied across the memristor.

In contrast, the voltage drop across the memristor increases and the current across it decreases while transitioning from a low resistance state (LRS) to a high resistance state (HRS), when an external negative voltage is applied across the memristor. Hence, the memristor device can function as a switch, and can transition from LRS to HRS (and vice versa). A memristor in the LRS allows current to flow across it, akin to a closed circuit or a turned-on switch. A memristor in the HRS allows no current to flow, similar to a turned-off switch.

The dynamics of memristors were further investigated and characterized using Lissajous figures by Chua and Kang [39]. A memristor's transitional behavior is identified as a pinched hysteresis loop, which underlies the special dynamics associated with a memristor. This makes it possible to explore the possibilities of using a memristor as an interesting nonlinear element for computation, even though our approach uses memristor only as a digital switch and a non-volatile memory element.

D. Applications of Memristors

Due to their non-volatile property, memristors and other ReRAMs are primarily employed as memory elements in novel memory architectures. A compelling design for caches has been proposed [40] that leverages 3D stacked ReRAMs. A framework for performing stateful logic using memristor-based nanoscale crossbar circuits has been investigated [41]. The crossbar fabric has also been shown to support parallel computation [42]. It has been demonstrated in [43] that Boolean logic operations can be reliably implemented using a single memristor – the trick lies in applying the input bits periodically in a sequential manner instead of applying them all at once. One of the impressive displays of using memristors at the core of parallel computation can be found in [25]. In [44], a promising solution has been suggested in order to unleash the potential of in-memory computation with memristors for big data applications. A search-based synthesis procedure has been used to design edge detection crossbars [45]. However, these designs are about 3.5 times larger than our designs, and no PSNR-based comparison of the resultant images has been performed in the aforementioned publication. Their approach directly uses our earlier work [37] but does not leverage model counting.

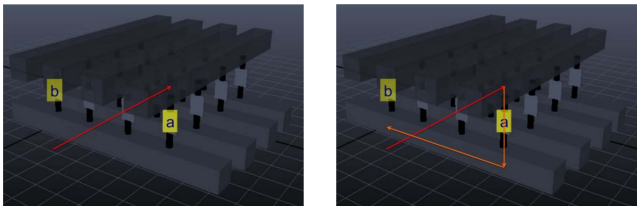
There exists exciting contemporary research in neuromorphic computing, both in the design of neuromorphic systems with non-volatile resistive components, as well as the development of new devices targeted towards neuromorphic architectures [46], [47]. A well-known technique for neuromorphic computation is the usage of spiking signals. A spiking based neuromorphic design was proposed in [48] that lead to more than 50% in energy savings with insignificant decrease in recognition probability. A system based on similar operating principles, in which both training and classification is performed on the crossbar array was proposed in [49]. It well known that neurons are dynamical systems; an alternative computational paradigm which takes advantage of such behavioral characteristics using Mott memristors was introduced in [50].

III. Flow-Based Computing With Crossbars

The key challenge in our flow-based computing approach [51] is to design the pattern in which data should be loaded onto a two-dimensional array of nanoscale memristors such that the flow of current through the crossbar can perform a desired computation. Figure 2 illustrates the flow-based computing approach using the simple Boolean formula "a AND b". All the memristors in the crossbar are first turned off; then, the data corresponding to the values of Boolean variables are loaded onto the memristors labeled "a" and "b" respectively. Finally, a flow of current introduced in one (say, rightmost) nanowire reaches another (say, leftmost) nanowire if and only if the formula "a AND b" is true.

In general, a Boolean formula is mapped onto a crossbar design with the following properties [51], [52]:

- Each memristor in the crossbar is mapped to a literal in the Boolean formula or the logical constants True and False.
- The memristors are switched ON or OFF depending on the patterns in which data is to be loaded for a specific computation and the specific input instance.
- A current flow is introduced in an *a priori* specified input nanowire.

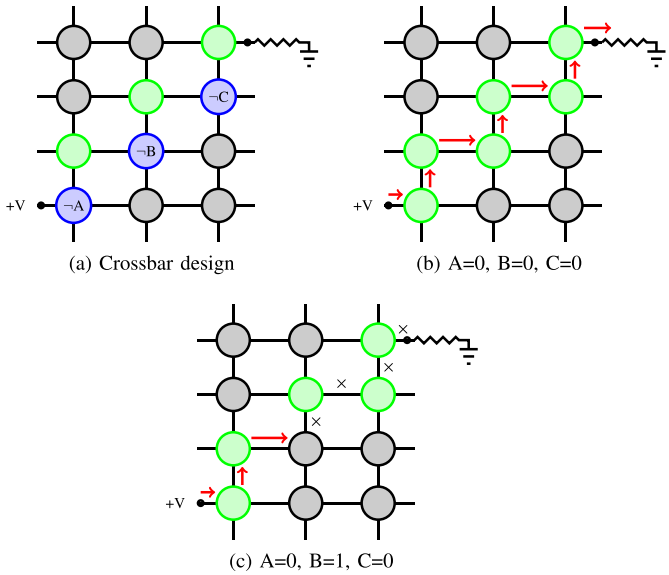


(a) Current flow introduced in the bottom row.

(b) Flow transition from row to column through ON memristor.

(c) Output flow observed at top row.

Fig. 2. Flow-based computing with crossbar circuits.



(a) Crossbar design

(b) A=0, B=0, C=0

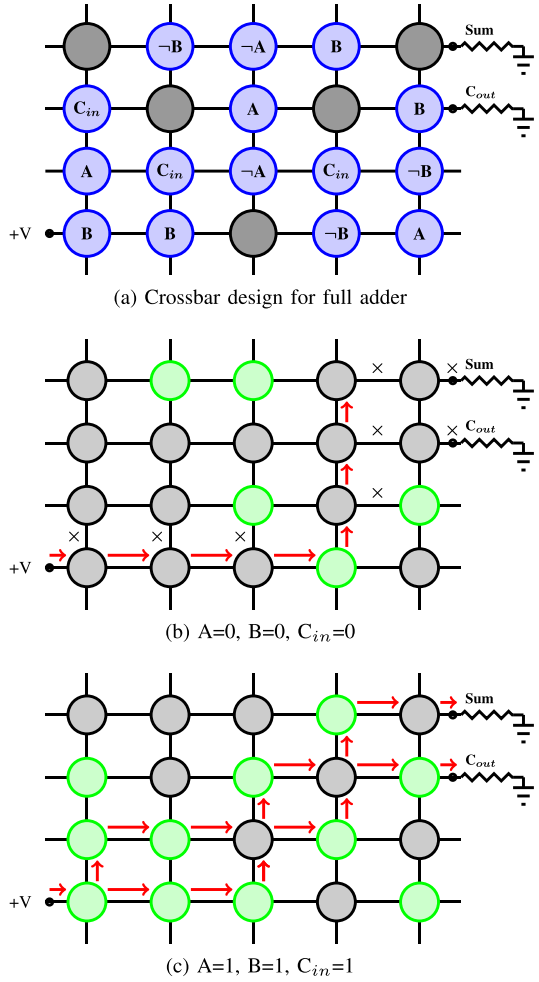
(c) A=0, B=1, C=0

Fig. 3. Crossbar implementing $\neg A \wedge \neg B \wedge \neg C$ with two input instances.

- If the Boolean formula evaluates to TRUE for a given input instance, then a current flow is observable at a predefined output nanowire. On the other hand, if the Boolean formula evaluates to FALSE for a given input instance, then a current flow is not observable at a predefined nanowire.

An example of a crossbar implementing the Boolean formula $\neg A \wedge \neg B \wedge \neg C$ is shown in Figure 3. The blue circles represent memristors which assume the value of literals, the green circles represent memristors in the ON state and the grey circles represent memristors in the OFF state. In Figure 3(b) and 3(c), the red arrows represent current flow and the crosses represent a lack of current flow.

As an illustration, Figure 4(a) shows the design of a 1-bit full adder that computes both the sum and the carry-out bits. A grey circle indicates a memristor that is always turned off; a blue circle is labeled with the literal whose value decides whether the corresponding memristor is turned on or off.



(a) Crossbar design for full adder

(b) A=0, B=0, C_{in}=0

(c) A=1, B=1, C_{in}=1

Fig. 4. Design of a memristor crossbar implementing a full adder with two input instances.

A memristor that is always turned on is indicated by a green circle.

Figure 4(b) shows the operation of the flow-based computing approach for the inputs $A = 0$, $B = 0$, and $C_{in} = 0$. The black circles indicate memristors that are turned off while the green memristors indicate memristors that are turned on. The red arrows and the black crosses indicate that there is no path that allows the current to flow from the input to the output – thereby, yielding a sum of 0 and a carry-out of 0.

Figure 4(c) shows the memristor crossbar when the data corresponding the inputs $A = 1$, $B = 1$, and $C_{in} = 1$ have been loaded into the nanoscale memristor crossbar. Again, the turned-on memristors are indicated by green circles and the turned-off memristors are shown by black circles. The red arrows show how the current can flow from the input to the outputs – thereby, yielding both a sum and a carry-out of 1.

In order to synthesize crossbar designs successfully, it is inefficient and cumbersome to stick to electrical models of crossbar circuits. Hence, we encapsulate crossbar circuits in a digital abstraction that can be easily manipulated using Boolean algebra to aid in the synthesis procedure:

Definition 1 (CROSSBAR): A memristor-based crossbar is a 3-tuple $\mathbb{C} = (\mathbb{M}, \mathbb{W}_r, \mathbb{W}_c)$ *where*

- $\mathbb{M} = \begin{pmatrix} m_{11} & m_{12} & \cdots & m_{1n} \\ \vdots & \vdots & \ddots & \vdots \\ m_{l1} & m_{l2} & \cdots & m_{ln} \end{pmatrix}$ *is a two-dimensional array of memristors with l rows and n columns, where $m_{ij} \in \{0, 1\}$ denotes the state of the memristor (ON or OFF) connecting row i with column j.*
- $\mathbb{W}_r = \{r_1, \cdots, r_l\}$ *is the set of horizontal nanowires such that the wire r_i provides the same input voltage to every memristor in row i.*
- $\mathbb{W}_c = \{c_1, \cdots, c_l\}$ *is the set of vertical nanowires such that the wire c_j provides the same input voltage to every memristor in column j.*

The memristor $m_{ij} = 0$ is said to be in the high-resistance state or OFF state and $m_{ij} = 1$ denotes a memristor in the low-resistance state or ON state.

Definition 2 (CROSSBAR DESIGN): A crossbar design $\mathcal{D}(\mathbb{M})$ maps each memristor m_{ij} in the crossbar $\mathbb{M}$ to one of the following: an input Boolean variable $v_1, \cdots, v_n$, its negations $\neg v_1, \cdots, \neg v_n$ or the logical constants True or False.

For a crossbar with l rows, n columns and v different input variables, each memristor can be mapped to $2v + 2$ different values. Hence the number of possible crossbar designs is as large as $(2v + 2)^{n \times l}$. As an illustration, the edge detection example in this manuscript has 16 Boolean variables on a 8×8 crossbar; this corresponds to a search space involving 1.158e+77 designs. The number of designs is comparable to the number of atoms in the known universe.

IV. Synthesis Using Model Counting

In order to search the exponentially growing space of possible crossbar designs, our synthesis procedure exploits a symbolic representation of Boolean functions and employs model counting [53] to guide a simulated annealing based search procedure. Our approach for synthesizing flow-based computing crossbar circuits relies on approximating the distance between a given crossbar design and the target Boolean function using model counting.

The Boolean satisfiability problem is regarded as one of the foundational problems in computer science. A propositional formula is deemed to be satisfiable if there exists an assignment of values to its variables for which the formula evaluates to true. By extension, model counting is the problem of computing the number of models for a given propositional formula, or finding the number of distinct input variable assignments for which the propositional formula evaluates to true. For a given crossbar design, we would want the output flow states to match the corresponding evaluations of the Boolean formula ϕ, for all possible input assignments. Hence, our objective is to count the number of models which satisfy the equivalence of the Boolean formula with the candidate crossbar design. This problem is readily solved using existing model counting algorithms. Extended discussions about approaches to model counting have been presented in a handbook [53].

Algorithm 1 summarizes our methodology. In line 1, we first pick a random crossbar design of size l rows and n columns. Each memristor in the design is mapped to either $True$, $False$, one of the variables $v_1, \ldots v_k$ or one of the negated variables $\neg v_1, \ldots, \neg v_k$, as sampled from a uniform random distribution.

Algorithm 1: Crossbar Synthesis Algorithm

Input : Target Boolean Formula ϕ over variables $\{v_1, v_2 \ldots v_k\}$
Size of crossbar $\mathbb{C}$: l rows and n columns
Initial temperature for simulated annealing T
Cooling rate c

Output : Crossbar Design $\mathcal{D}(\mathbb{M})$ mapping each memristor $m_{ij} \in \mathbb{M}$ to the set $\{True, False, v_1, \ldots, v_k, \neg v_1, \ldots, \neg v_k\}$

1 $\mathcal{D}_1 \leftarrow$ PickRandomCrossbarDesign$(l, n, v_1, \ldots v_k)$
$\mathcal{B}(\mathcal{D}_1) \leftarrow$ BooleanFlow$(\mathcal{D}_1)$
$\Delta_1 \leftarrow$ ModelCount$(\mathcal{B}(\mathcal{D}_1) \oplus \phi)$
while $\Delta_i > 0$ **do**
2 $\quad \mathcal{D}_{i+1} \leftarrow$ PerturbCrossbarDesign$(\mathcal{D}_i, \phi)$
$\quad \mathcal{B}(\mathcal{D}_{i+1}) \leftarrow$ BooleanFlow$(\mathcal{D}_{i+1})$
$\quad \Delta_{i+1} \leftarrow$ ModelCount$(\mathcal{B}(\mathcal{D}_{i+1}) \oplus \phi)$
$\quad$ **if** $rand(0, 1) < e^{-(\Delta_{i+1} - \Delta_i)/T}$ **then**
3 $\quad\quad i \leftarrow i + 1$
4 $\quad$ **end**
5 $\quad T \leftarrow c \times T$
6 **end**
7 **Return** crossbar design $\mathcal{D}_i$

For each crossbar design $\mathcal{D}$, we assume that a flow of current is injected into the top horizontal nanowire. Then, we compute the number of assignments to the Boolean variables which cause the flow of current to reach the lowermost horizontal nanowire of the crossbar design $\mathcal{D}$.

In order to evaluate the model, we must first compute the Boolean formula realized by the crossbar. Let $r_i^{(t)}$ denote the flow value of row i at time t, and $c_j^{(t)}$ denote the flow value of column j at time t. At $t = 0$, only the first row has flow, i.e., $r_1^{(0)} = True$ and all other rows and columns are set to $False$. For all $t > 0$, we define the following transitions for each nanowire based on the ability of turned-on memristors to create a path between their horizontal and vertical nanowires. In this way, we symbolically compute the Boolean formula representing the values of the memristors for which a flow reaches the topmost nanowire of the crossbar and denote it by BooleanFlow$(\mathcal{D}_1)$ in line 2 of the algorithm.

$$\forall i \in \{2, \ldots, n\}, r_i^{(t+1)} \iff \left(r_i^{(t)} \vee \bigvee_{1 \le j \le n} (m_{ij} \wedge c_j^{(t)})\right)$$

$$\forall j \in \{1, \ldots, m\}, c_j^{(t+1)} \iff \left(c_j^{(t)} \vee \bigvee_{1 \le i \le l} (m_{ij} \wedge r_i^{(t)})\right)$$

The above transitions of the rows and columns in the crossbar have been represented using Boolean functions and can be described succinctly using Boolean Decision Diagrams (BDDs), And-Inverter-Graphs (AIGs) or other representations.

Line 3 of the algorithm computes the number of satisfiable instances Δ_1, which corresponds to the symmetric difference between the target Boolean formula ϕ and the formula corresponding to the computation performed by the

TABLE I

COMPUTATION DELAY (PICOSECONDS) TO COMPUTE THE MSB OF n-BIT ADDITION FOR AREA-OPTIMIZED CROSSBARS

#bits / input	BDD [16]	MIG -IMP [54]	MIG -MAJ [54]	Model counting	Speedup
2	425	3400	1020	340	125%
3	765	5100	1530	340	225%
4	1020	6800	2040	425	**240%**

TABLE II

POWER (IN μW) REQUIRED TO COMPUTE THE MSB OF BINARY ADDITION FOR AREA-OPTIMIZED CROSSBARS

#bits / input	BDD [16]	MIG -IMP [54]	MIG -MAJ [54]	Model counting	Improvement
2	240	1200	360	300	80%
3	420	1800	540	390	**107.7%**
4	600	2400	720	720	83.3%

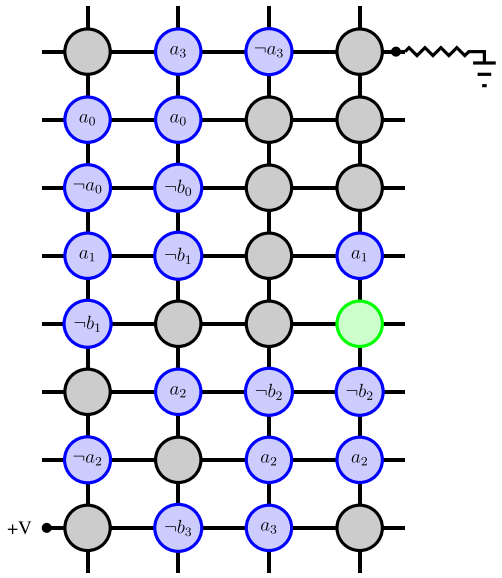
crossbar design $\mathcal{D}_1$. Several competitive implementations of approximate model counting algorithms are available and our approach is agnostic to the choice of the model counting strategy as long as the algorithm produces the count of 0 feasible models only for unsatisfiable formula [53].

The loop in line 4 through line 12 continues to modify the current crossbar design, evaluate the function computed by this perturbed design, and count the number of satisfiable instances to the Boolean formula corresponding to the symmetric difference of the target Boolean formula and the formula computed by the current crossbar design. Lines 8 through 10 show the probabilistic acceptance step of the simulated annealing algorithm. New crossbar designs are always accepted if they are better than the existing crossbar design. New crossbar designs that are worse than the existing crossbar design are accepted with a probability that is a function of both the quality of the designs and the current temperature of the simulated annealing algorithm.

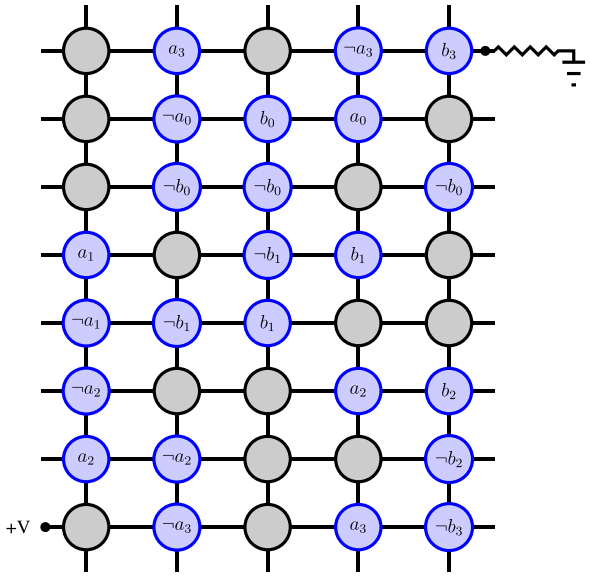
At every iteration of the loop, the temperature of the simulated annealing algorithm is slightly lowered. When the number of satisfiable instances for the symmetric difference becomes zero (or falls below an assigned threshold), our algorithm stops and reports the synthesized crossbar design.

The compact crossbar designs for a 4-bit addition and a 4-bit comparator obtained from model counting are shown in Figure 5. Both the crossbars compute the most significant bit of n-bit binary computations. The correctness of the designs has also been verified using ngSPICE-26 simulations [55]. Table I compares the crossbar designed using model counting with those designed using other competing approaches. The last column in Table I represents the speedup achieved through model counting with respect to the prior best method.

Similarly, Table II presents a comparison between the power required by crossbars designed using our model counting approach and compares it to memristor-based designs obtained using other approaches. The last column in Table II represents the power consumption ratio of crossbars produced through model counting with respect to the prior best method.



(a) Comparator for 4-bit binary comparison.



(b) MSB computation for 4-bit binary addition.

Fig. 5. Compact crossbars synthesized using model counting.

V. EDGE DETECTION WITH FLOW-BASED INPUT-AWARE CROSSBAR COMPUTING

Applied problems in domains such as image processing and vision require reasonably accurate but not necessarily exact computations. Such soft applications naturally provide an opportunity for the exploration of approximate input-aware computing with a focus on correctness over the likely inputs as opposed to all possible inputs.

One of the fundamental procedures in image processing is edge detection, that is, the demarcation of boundaries between the different objects in a given image. While there exist numerous algorithms to perform efficient and highly sophisticated edge detection in images [56], we can say that an "edge" exists if the difference between the gray-scale values of two pixels exceeds a predefined threshold value. Since a pixel can have any value between 0 and 255, the simplest edge

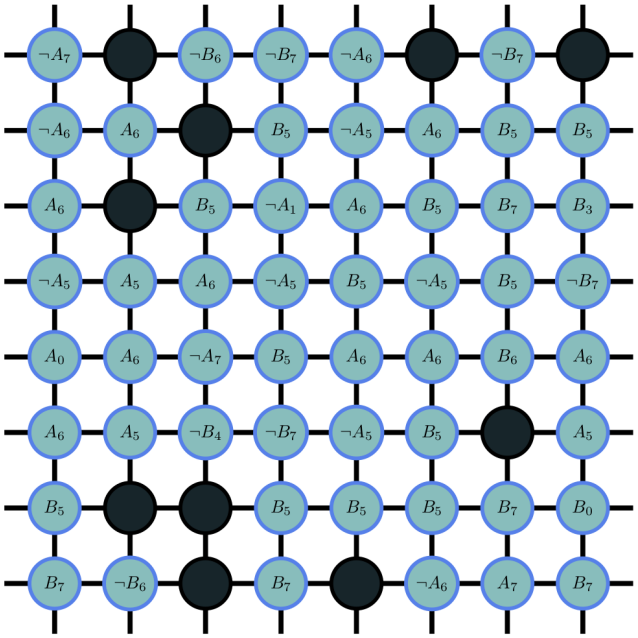
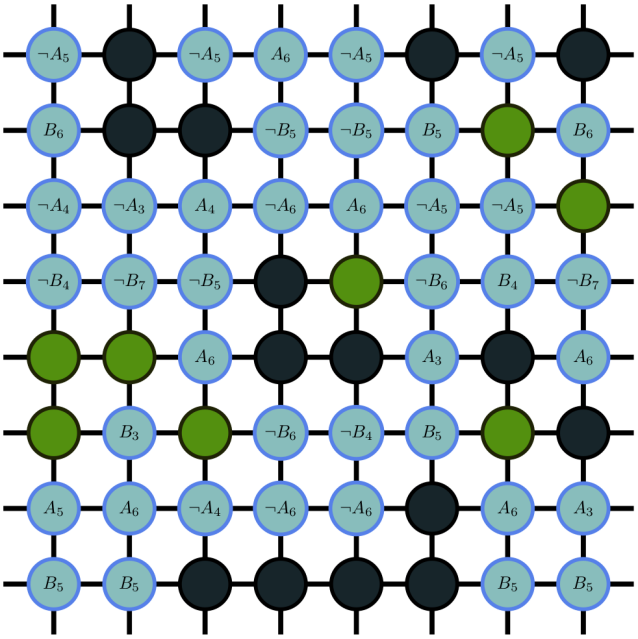
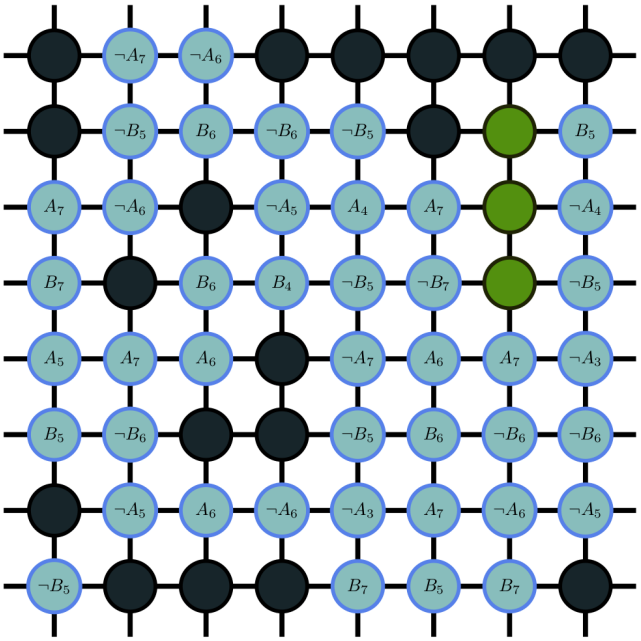


Fig. 6. Crossbar design based on all possible inputs, median PSNR = 2.4dB.



(a) $\mathrm{XBAR_A}$, median PSNR = 6.275 dB.



(b) $\mathrm{XBAR_B}$, median PSNR = 5.388 dB.

Fig. 7. Crossbars for edge detection on input-aware pixel pairs.

detection operation involves computing the difference between two 8-bit binary numbers followed by a comparison of the difference with a constant 8-bit binary number representing the threshold at which an edge is said to exist.

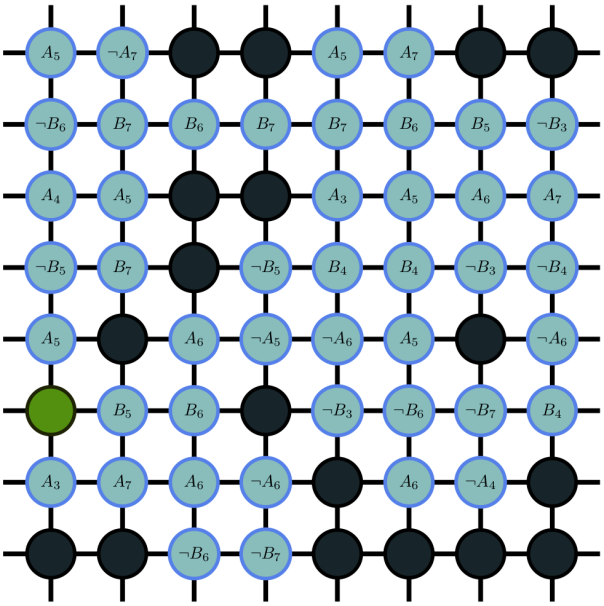
We employ the model counting approach explained in the previous section to synthesize crossbars for approximate computation of Boolean formulas. In this setting of approximate computation, we minimize the symmetric difference between the threshold-based exact edge detection and the Boolean function computed by the synthesized crossbar circuit. The approximate detection of edges is sufficient for practical purposes, and we do not require the number of satisfiable instances for the symmetric difference to reach zero – a fairly low value close to zero is acceptable.

Since computation is performed on two 8-bit pixels for threshold-based edge detection, the truth table consists of 65536 entries. Employing our model counting approach, we are able to obtain a crossbar design which has an accuracy of $\sim$85% over all possible input values.

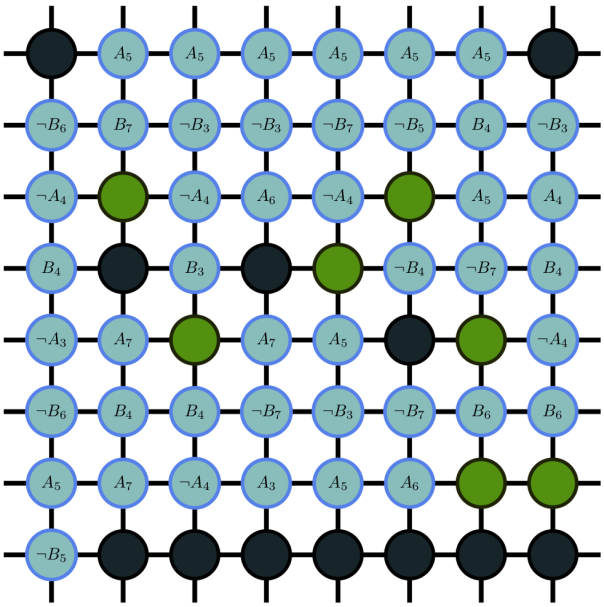
Figure 6 illustrates the approximate crossbar design. In the crossbar image, the dark circles represent permanently OFF memristors, the green unlabeled circles represent permanently ON memristors, and the light blue circles represent memristors whose values correspond to the literals. The variables $A_0, A_1, \ldots, A_7$ and $B_0, B_1, \ldots, B_7$ represent the bits of the two pixels, and the variables $\neg A_0, \neg A_1, \ldots, \neg A_7$ and $\neg B_0, \neg B_1, \ldots, \neg B_7$ represent their respective negations.

We employ the BSD500 database [57] for testing the performance of the obtained crossbar. The peak signal to noise ratio (PSNR) metric is commonly used for quality assessment between two perceptually equivalent images [58]. The PSNR between the edge images produced by exact computation and those produced by the approximate crossbar provides an estimate of the quality of edge detection performed by the crossbar circuit. The synthesized crossbar design is provided in Figure 6 and the histogram of the PSNR for all images in BSD500 are presented in Figure 9.

The PSNR histogram in Figure 6 shows that a single approximate computing crossbar over all the truth table entries does not really perform well in the case of edge detection since not all pixel pairs occur with the same frequency. Hence, we identify a subset of all possible pixel pairs that occur with high frequency. A given pixel pair belongs in this subset if it occurs more than three times in each image, on an average, over all the images in BSD500. We then design approximate input-aware edge-detection crossbars with a focus on these likely inputs. A few crossbar designs obtained using this approach are illustrated in Figures 7 and 8.



(a) $XBAR_C$, median PSNR = 5.8 dB.

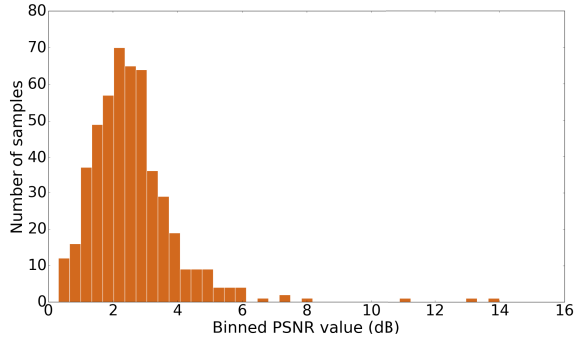


(b) $XBAR_D$, median PSNR = 6 dB.

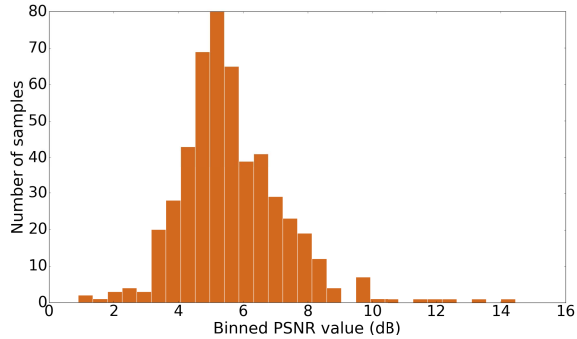
Fig. 8. Crossbars for edge detection on input-aware pixel pairs.

The downsized truth table involves only the chosen subset of pixel pairs and contains 16,658 entries. This input-aware truth-table enables the synthesized crossbar designs to perform well on the relevant inputs even though they have relatively lower accuracy over all possible truth table inputs. The subset of pixel pairs is utilized to build a model for the model counting approach, in order to synthesize a family of crossbar circuits for input-aware approximate edge detection.

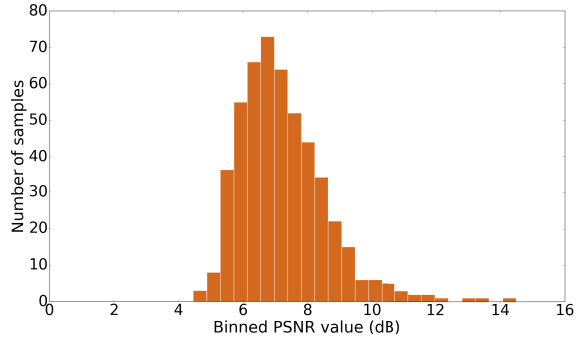
Since the standalone designs are not accurate, it is intuitive that they may not agree on the same mistakes. Given a family of crossbar designs approximately computing the same function, their outputs should be combined by using a majority operation. In order to test this hypothesis, we have synthesized



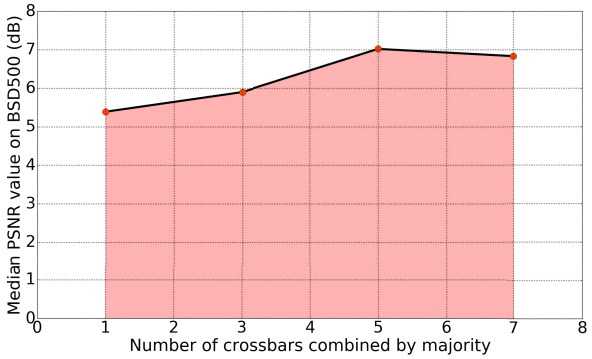
(a) Histogram of PSNR values for outputs produced by the crossbar design synthesized on all possible inputs, median PSNR = 2.4dB



(b) Histogram of PSNR values for outputs produced by a single input-aware crossbar design, median PSNR = 5.38dB



(c) Histogram of PSNR values for outputs produced by majority-based combination of five input-aware crossbar designs, median PSNR = 7.02dB.



(d) Median PSNR values for majority-based combination of input-aware crossbar outputs.

Fig. 9. Histogram of PSNR values corresponding to output images produced by majority-based combination of crossbars. A combination of crossbars produce a better (i.e. shifted to the right) PSNR distribution that a single input-aware crossbar.

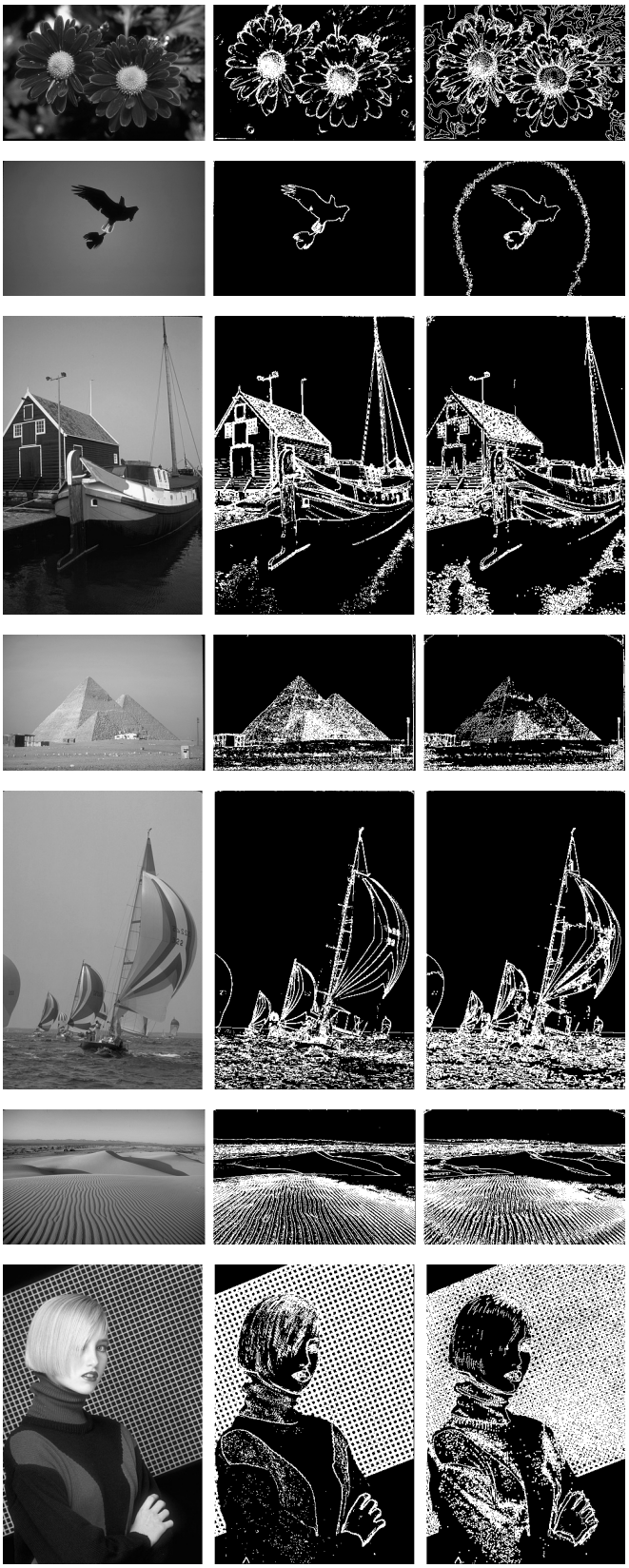


Fig. 10. The input gray-scale image, the computed edge image and the output edge image obtained via majority-based combination of approximately correct input-aware crossbar outputs, respectively.

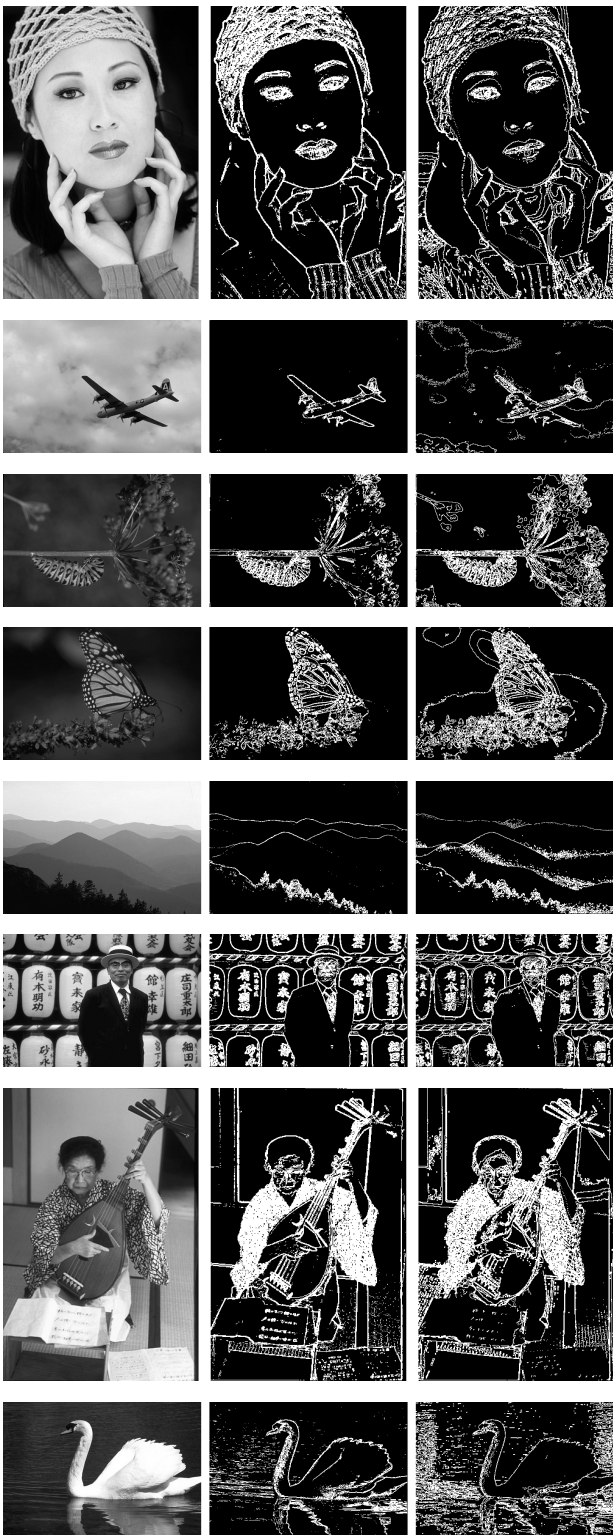


Fig. 11. The input grayscale image, the computed edge image and the edge image based on combining approximately correct input-aware crossbar outputs using a majority function.

a number of different crossbar designs with varying degrees of accuracy. The PSNR values between the computed edge images and the output images obtained by majority-based combination are presented in Figure 9. The input-aware crossbar is better i.e. it has a higher PSNR than a crossbar designed using all possible inputs. Similarly, a family of 5 crossbars produces better PSNR values than any crossbar in the family. While the best crossbar has a performance of about 6.275dB, combining it with crossbars of even less performance produces

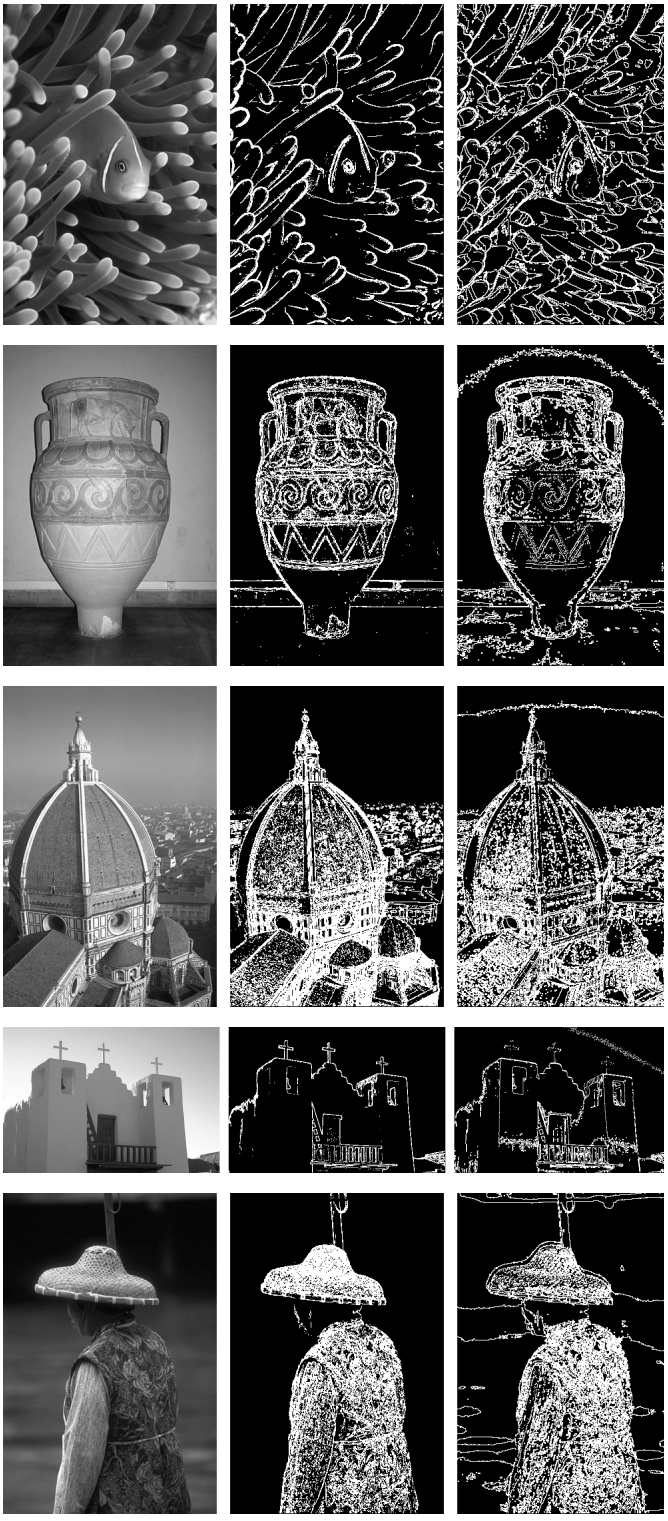


Fig. 12. The input grayscale image, the computed edge image and the edge image based on combining approximately correct input-aware crossbar outputs using a majority function.

an overall design with a PSNR of 7.02dB. This shows the importance of using a (small) family of approximate designs as opposed to the best approximate design.

The edge images produced by the combination of approximately-correct crossbars are better than the edge images produced by a single crossbar with higher accuracy over all possible inputs. The input gray-scale images, the computed edge images and the output edge images produced by combining the outputs from a family of five approximate input-aware crossbars are presented in Figures 10, 11 and 12.

VI. Conclusion and Future Work

We have explored the design of input-aware approximately-correct flow-based crossbar circuits for basic computer vision applications, like edge detection. Our proposed approach is driven by model counting to explore the design space of crossbar circuits. The synthesized designs are compact, and approximately compute the necessary functions for the desired applications. We have produced an assortment of 8×8 crossbar designs belonging to two different operational groups. One of the groups can perform threshold-based edge detection with high accuracy on all possible pixel pairs and the other group performs approximate edge detection on an application-specific subset of input values. In case of the second set of crossbar designs, the outputs from individual crossbars are combined using the majority function to yield the final output image. We have tested the performance of our method on the well-known BSD500 database. We utilize the peak-signal-to-noise-ratio (PSNR) metric to evaluate the quality of the output images. The results obtained from the input-aware approximate computation are observably better than those produced by the more accurate general-purpose crossbars.

Our work has focused on the design of crossbar arrays for approximate computations and has not explored the impact of peripheral circuitry. Our approach only requires standard peripheral circuitry required to read and write data on the memristor crossbar but does not need any additional peripheral circuitry to control the flows. Our design automation approach is based on the native use of memristor crossbars as a computational fabric without trying to control the flow of information through the crossbar using external peripheral circuitry. In fact, our design permits all possible flows to occur and the memristor crossbar is designed in a manner where all possible flows put together enable a desired computation. In previous experimental studies [59] we have shown that a 1-bit adder can be implemented in practice on a hardware memristor crossbar and the current theoretical investigation extends the same approach without requiring any additional changes in peripheral circuitry. It should be noted that the efficiency and correctness of the peripheral circuitry will have an impact on the overall performance of our approach, and these important impacts need to be investigated.

The design of approximate input-aware application-specific solutions using flow-based computing is an area that is ripe with exciting opportunities. Future work related to the approach presented in this manuscript includes, but is not limited to, the following possibilities:

1. A natural question to investigate is the dependency of the crossbar design on the data set used to train our input-aware approach. Figure 13 illustrates the BSD500 and CIFAR [60] datasets. The axes in the color map represent the intensity of the pixels in the pair and the color bar represents the natural logarithm of the frequency with which the pixel pairs occur in the respective datasets.

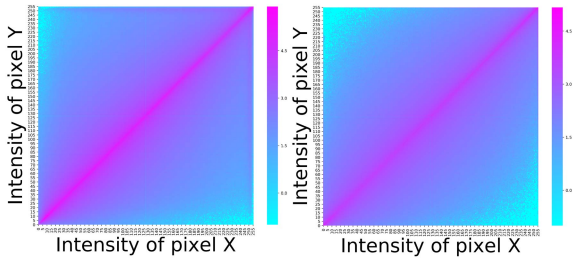


Fig. 13. Pixel pair distributions in BSD500 and CIFAR.

Because of the remarkable visual similarity of these distributions, one can conjecture that the crossbar design is not sensitive to the choice of a specific data set. Future work is needed to investigate this dependence more thoroughly.

2. The exploration of the design space of crossbars can benefit by leveraging the massively parallel multi-core computing capabilities of general purpose graphics processing units (GPGPUs). In particular, deep learning approaches may be helpful in accelerating the evaluation of a candidate crossbar design.
3. A number of higher level artificial intelligence applications depend on fundamental operations like convolutions, clustering and classification. Embedding such operations on compact crossbar circuits may have a significant impact on designing hardware accelerators for machine learning and artificial intelligence.
4. Our current approach has only employed memristors as binary switches. Memristors and other resistive memory devices exhibit a multitude of discrete memory states in the interval between the on and off states. This interesting property can unlock a novel pathway for doing ternary and other n-ary logic computations on the memristor crossbar fabric.

REFERENCES

[1] P. J. Kuekes, R. S. Williams, and J. R. Heath, "Molecular wire crossbar memory," U.S. Patent 6 128 214 A, Oct. 3, 2000.
[2] Y. Chen and R. S. Williams, "Configurable nanoscale crossbar electronic circuits made by electrochemical reaction," U.S. Patent 6 518 156 B1, Feb. 11, 2003.
[3] C. Li *et al.*, "Large Memristor crossbars for analog computing," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, May 2018, pp. 1–4.
[4] C. Liu *et al.*, "A Memristor crossbar based computing engine optimized for high speed and accuracy," in *Proc. IEEE Comput. Soc. Annu. Symp. VLSI (ISVLSI)*, Jul. 2016, pp. 110–115.
[5] N. A. Melosh *et al.*, "Ultrahigh-density nanowire lattices and circuits," *Science*, vol. 300, no. 5616, pp. 112–115, 2003.
[6] G.-Y. Jung *et al.*, "Circuit fabrication at 17 nm half-pitch by nanoimprint lithography," *Nano Lett.*, vol. 6, no. 3, pp. 351–354, 2006.
[7] B. Govoreanu *et al.*, "10×10 nm^{2}Hf/HfOx crossbar resistive RAM with excellent performance, reliability and low-energy operation," in *IEDM Tech. Dig.*, Dec. 2011, pp. 31.6.1–31.6.4.
[8] K.-S. Li *et al.*, "Utilizing sub-5 nm sidewall electrode technology for atomic-scale resistive memory fabrication," in *Symp. VLSI Technol. Dig. Tech. Papers*, Jun. 2014, pp. 1–2.
[9] S. Lee, J. Sohn, Z. Jiang, H.-Y. Chen, and H.-S. P. Wong, "Metal oxide-resistive memory using graphene-edge electrodes," *Nature Commun.*, vol. 6, p. 8407, Sep. 2015.
[10] S. Pi *et al.*, "Memristor crossbar arrays with 6-nm half-pitch and 2-nm critical dimension," *Nature Nanotechnol.*, vol. 14, no. 1, pp. 35–39, 2019.
[11] M. A. Rothman, V. J. Zimmer, G. P. Mudusuru, J. Yao, and J. Lin, "Technology to facilitate rapid booting with high-speed and low-speed nonvolatile memory," U.S. Patent 15 484 513, Oct. 11, 2018.
[12] F. T. Hady, A. Foong, B. Veal, and D. Williams, "Platform storage performance with 3D XPoint technology," *Proc. IEEE*, vol. 105, no. 9, pp. 1822–1833, Sep. 2017.
[13] S. H. Jo, T. Kumar, S. Narayanan, W. D. Lu, and H. Nazarian, "3D-stackable crossbar resistive memory based on field assisted superlinear threshold (FAST) selector," in *IEDM Tech. Dig.*, Dec. 2014, pp. 6.7.1–6.7.4.
[14] M. Rothman and V. Zimmer, "Low latency boot from zero-power state," U.S. Patent 15 891 124, Feb. 7, 2019.
[15] A. Velasquez and S. K. Jha, "Automated synthesis of crossbars for nanoscale computing using formal methods," in *Proc. IEEE/ACM Int. Symp. Nanosc. Archit.*, Jul. 2015, pp. 130–136.
[16] D. Chakraborty and S. K. Jha, "Automated synthesis of compact crossbars for sneak-path based in-memory computing," in *Proc. Design, Automat. Test Eur. Conf. Exhib.*, Mar. 2017, pp. 770–775.
[17] D. Chakraborty and S. K. Jha, "Design of compact memristive in-memory computing systems using model counting," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, May 2017, pp. 1–4.
[18] G. E. Moore, "Cramming more components onto integrated circuits," *Proc. IEEE*, vol. 86, no. 1, pp. 82–85, Jan. 1998.
[19] G. E. Moore, "Progress in digital integrated electronics," in *Proc. Int. Electron Devices Meeting*, vol. 21, 1975, pp. 11–13.
[20] G. Baccarani, M. R. Wordeman, and R. H. Dennard, "Generalized scaling theory and its application to a 1/4 micrometer MOSFET design," *IEEE Trans. Electron Devices*, vol. ED-31, no. 4, pp. 452–462, Apr. 1984.
[21] X. Huang *et al.*, "Sub-50 nm p-channel FinFET," *IEEE Trans. Electron Devices*, vol. 48, no. 5, pp. 880–886, May 2001.
[22] A. W. Topol *et al.*, "Three-dimensional integrated circuits," *IBM J. Res. Develop.*, vol. 50, nos. 4–5, pp. 491–506, Jul./Sep. 2006.
[23] R. G. Dreslinski, M. Wieckowski, D. Blaauw, D. Sylvester, and T. Mudge, "Near-threshold computing: Reclaiming Moore's law through energy efficient integrated circuits," *Proc. IEEE*, vol. 98, no. 2, pp. 253–266, Feb. 2010.
[24] C. Li *et al.*, "Analogue signal and image processing with large memristor crossbars," *Nature Electron.*, vol. 1, no. 1, pp. 52–59, 2018.
[25] A. Haron, J. Yu, R. Nane, M. Taouil, S. Hamdioui, and K. Bertels, "Parallel matrix multiplication on memristor-based computation-in-memory architecture," in *Proc. Int. Conf. High Perform. Comput. Simulation (HPCS)*, Jul. 2016, pp. 759–766.
[26] E. J. Merced-Grafals, N. Dávila, N. Ge, R. S. Williams, and J. P. Strachan, "Repeatable, accurate, and high speed multi-level programming of memristor 1T1R arrays for power efficient analog computing applications," *Nanotechnology*, vol. 27, no. 36, 2016, Art. no. 365202.
[27] J. von Neumann, "First draft of a report on the EDVAC," *IEEE Ann. Hist. Comput.*, vol. 15, no. 4, pp. 27–75, 1993.
[28] A. W. Burks, H. H. Goldstine, and J. von Neumann, "Preliminary discussion of the logical design of an electronic computing instrument," in *The Origins of Digital Computers*. Springer, 1982, pp. 399–413.
[29] J. Backus, "Can programming be liberated from the von Neumann style?: A functional style and its algebra of programs," *Commun. ACM* vol. 21, no. 8, pp. 613–641, 1978.
[30] D. Patterson *et al.*, "A case for intelligent RAM," *IEEE Micro*, vol. 17, no. 2, pp. 34–44, Mar. 1997.
[31] W. A. Wulf and S. A. McKee, "Hitting the memory wall: Implications of the obvious," *ACM SIGARCH Comput. Archit. News*, vol. 23, no. 1, pp. 20–24, Mar. 1995.
[32] A. J. Smith, "Cache memories," *ACM Comput. Surv.*, vol. 14, no. 3, pp. 473–530, 1982.
[33] J. B. Dennis and D. P. Misunas, "A preliminary architecture for a basic data-flow processor," *ACM SIGARCH Comput. Archit. News*, vol. 3, no. 4, pp. 126–132, Jan. 1975
[34] J. Ahn, S. Yoo, O. Mutlu, and K. Choi, "PIM-enabled instructions: A low-overhead, locality-aware processing-in-memory architecture," in *Proc. ACM/IEEE 42nd Annu. Int. Symp. Comput. Archit. (ISCA)*, Jun. 2015, pp. 336–348.
[35] Q. Zhu, K. Vaidyanathan, O. Shacham, M. Horowitz, L. Pileggi, and F. Franchetti, "Design automation framework for application-specific logic-in-memory blocks," in *Proc. 23rd Int. Conf. Appl.-Specific Syst. Archit. Processors*, Jul. 2012, pp. 125–132.

[36] H. A. Du Nguyen, L. Xie, M. Taouil, R. Nane, S. Hamdioui, and K. Bertels, "Computation-in-memory based parallel adder," in *Proc. IEEE/ACM Int. Symp. Nanosc. Archit.*, Jul. 2015, pp. 57–62.
[37] D. Chakraborty, S. Raj, J. C. Gutierrez, T. Thomas, and S. K. Jha, "In-memory execution of compute kernels using flow-based memristive crossbar computing," in *Proc. IEEE Int. Conf. Rebooting Comput. (ICRC)*, Nov. 2017, pp. 1–6.
[38] L. O. Chua, "Memristor-the missing circuit element," *IEEE Trans. Circuit Theory*, vol. CT-18, no. 5, pp. 507–519, Sep. 1971.
[39] L. O. Chua and S. M. Kang, "Memristive devices and systems," *Proc. IEEE*, vol. 64, no. 2, pp. 209–223, Feb. 1976.
[40] D. L. Lewis and H.-H. S. Lee, "Architectural evaluation of 3D stacked RRAM caches," in *Proc. IEEE Int. Conf. 3D Syst. Integr.*, Sep. 2009, pp. 1–4.
[41] X. Zhu, X. Yang, C. Wu, N. Xiao, and X. Yi, "Performing stateful logic on memristor memory," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 60, no. 10, pp. 682–686, Oct. 2013.
[42] Y. V. Pershin and M. Di Ventra, "Solving mazes with memristors: A massively parallel approach," *Phys. Rev. E, Covering Stat. Nonlinear Biol. Soft Matter Phys.*, vol. 84, no. 4, Oct. 2011, Art. no. 046703.
[43] E. Gale, B. de L. Costello, and A. Adamatzky, "Boolean logic gates from a single memristor via low-level sequential logic," in *Proc. Int. Conf. Unconventional Comput. Natural Comput.* Berlin, Germany: Springer, 2013, pp. 79–89.
[44] S. Hamdioui *et al.*, "Memristor based computation-in-memory architecture for data-intensive applications," in *Proc. Design, Automat. Test Eur. Conf. Exhib.*, 2015, pp. 1718–1725.
[45] S. Khokhar and A. Khalid, "Nanoscale memristive crossbar circuits for approximate edge detection in smart cameras," in *Proc. 9th Annu. Inf. Technol. Electron. Mobile Commun. Conf. (IEMCON)*, Nov. 2018, pp. 749–754.
[46] B. Liu *et al.*, "Programmable synaptic metaplasticity and below femtojoule spiking energy realized in graphene-based neuromorphic memristor," *ACS Appl. Mater. Interfaces*, vol. 10, no. 24, pp. 20237–20243, 2018.
[47] Z. Wang *et al.*, "Memristors with diffusive dynamics as synaptic emulators for neuromorphic computing," *Nature Mater.*, vol. 16, no. 1, pp. 101–108, 2017.
[48] C. Liu *et al.*, "A spiking neuromorphic design with resistive crossbar," in *Proc. 52nd ACM/EDAC/IEEE Design Automat. Conf. (DAC)*, Jun. 2015, pp. 1–6,
[49] A. M. Hassan, C. Yang, C. Liu, H. H. Li, and Y. Chen, "Hybrid spiking-based multi-layered self-learning neuromorphic system based on memristor crossbar arrays," in *Proc. Design, Automat. Test Eur. Conf. Exhib. (DATE)*, Mar. 2017, pp. 776–781.
[50] B. Yan, X. Cao, and H. H. Li, "A neuromorphic design using chaotic mott memristor with relaxation oscillation," in *Proc. 55th Annu. Design Automat. Conf.*, 2018, p. 167.
[51] S. K. Jha, D. E. Rodriguez, J. E. Van Nostrand, and A. Velasquez, "Computation of Boolean formulas using sneak paths in crossbar computing," U. S. Patent 9 319 047 B2, Apr. 19, 2016.
[52] A. Velasquez, "Computation Boolean formulas using sneak paths crossbar computing," Ph.D. dissertation, Dept. Comput. Sci., Univ. Central Florida, Orlando, FL, USA, 2014.
[53] A. Biere, M. Heule, and H. van Maaren, *Handbook of Satisfiability*, vol. 185. Amsterdam, The Netherlands: IOS Press, 2009.
[54] S. Shirinzadeh, M. Soeken, P.-E. Gaillardon, and R. Drechsler, "Fast logic synthesis for RRAM-based in-memory computing using majority-inverter graphs," in *Proc. Conf. Design Automat. Test Eur.*, Mar. 2016, pp. 948–953.
[55] P. Nenzi and H. Vogt, "Ngspice circuit simulator release 26," Tech. Rep., 2014.
[56] S. Bhardwaj and A. Mittal, "A survey on various edge detector techniques," *Procedia Technol.*, vol. 4, pp. 220–226, Jan. 2012.
[57] D. Martin, C. Fowlkes, D. Tal, and J. Malik, "A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics," in *Proc. 8th IEEE Int. Conf. Comput. Vis. (ICCV)*, vol. 2, Jul. 2001, pp. 416–423.
[58] Z. Wang, H. R. Sheikh, and A. C. Bovik, "No-reference perceptual quality assessment of JPEG compressed images," in *Proc. Int. Conf. Image Process.*, vol. 1, Sep. 2002, p. I.
[59] Z. Alamgir, K. Beckmann, N. Cady, A. Velasquez, and S. K. Jha, "Flow-based computing on nanoscale crossbars: Design and implementation of full adders," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, May 2016, pp. 1870–1873.
[60] A. Krizhevsky *et al.*, "Learning multiple layers of features from tiny images," Univ. Toronto Scarborough, Toronto, ON, Canada, Tech. Rep., 2009, vol. 1, no. 4, p. 7.

Dwaipayan Chakraborty received the Ph.D. degree in computer science from the University of Central Florida, Orlando, FL, USA, in 2019. His research interests include post Moore's Law computer architectures, memristor-based computing, stochastic computing, and nanoscale architectures. He has published papers in several reputed international conferences and journal, including the Design, Automation, and Test in Europe (DATE 2016, 2017), the IEEE International Symposium on Circuits and Systems (ISCAS 2017), the IEEE International Conference on Nanotechnology (IEEE NANO 2017), and the IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS (TCAS) II.

Sunny Raj received the B.Tech. degree in computer science and engineering from IIT (Indian School of Mines), Dhanbad, India, in 2011. He is currently pursuing the Ph.D. degree in computer science with the University of Central Florida, Orlando, FL, USA. His research interests include machine learning, quantum computing, and emerging computer architectures. He has published papers at *BMC Bioinformatics Journal* and reputed international conferences, including the 16th International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS-2018), the 10th International Symposium on Foundations and Practice of Security (FPS 2017), and the IEEE International Conference on Nanotechnology (IEEE NANO 2017).

Steven Lawrence Fernandes received the Ph.D. degree in electronics and communication engineering from the Karunya Institute of Technology and Sciences, India, in 2017. He was a Post-Doctoral Researcher in electrical and computer engineering with the University of Alabama at Birmingham, USA. He is currently a Post-Doctoral Researcher in computer science with the University of Central Florida, Orlando, FL, USA. He has published papers at *Pattern Recognition Letters* and *Computers in Biology and Medicine*. His research interests include machine learning and computer vision. He has served as a Guest Editor of the *Journal of Computational Science* (2016–2017) and *Future Generation Computer Systems* (2017–2018).

Sumit Kumar Jha received the Ph.D. degree in computer science from Carnegie Mellon University in 2010. He was with Microsoft Research India, General Motors, INRIA France, involved in R&D problems and the Air Force Research Lab Information Directorate. He is currently an Associate Professor with the Computer Science Department, University of Central Florida, Orlando, FL, USA. His research has been supported by the National Science Foundation, the Air Force Office of Scientific Research, the Oak Ridge National Laboratory, the Royal Bank of Canada, the Florida Center for Cybersecurity, and the Air Force Research Laboratory. Dr. Jha received the prestigious Air Force Young Investigator Award in 2016, and his research has led to three Best Paper Awards.