A Native Stochastic Computing Architecture Enabled by Memristors

Phil Knag, *Student Member, IEEE*, Wei Lu, *Member, IEEE*, and Zhengya Zhang, *Member, IEEE*

***Abstract*—A two-terminal memristor device is a promising digital memory for its high integration density, substantially lower energy consumption compared to CMOS, and scalability below 10 nm. However, a nanoscale memristor is an inherently stochastic device, and extra energy and latency are required to make a deterministic memory based on memristors. Instead of enforcing deterministic storage, we take advantage of the nondeterministic memory for native stochastic computing, where the randomness required by stochastic computing is intrinsic to the devices without resorting to expensive stochastic number generation. This native stochastic computing system can be implemented as a hybrid integration of memristor memory and simple CMOS stochastic computing circuits. We use an approach called group write to program memristor memory cells in arrays to generate random bit streams for stochastic computing. Three methods are proposed to program memristors using stochastic bit streams and compensate for the nonlinear memristor write function: voltage predistortion, parallel single-pulse write, and downscaled write and upscaled read. To evaluate these technical approaches, we show by simulation a memristor-based stochastic processor for gradient descent optimization, and *k*-means clustering. The native stochastic computing based on memristors demonstrates key advantages in energy and speed in compute-intensive, data-intensive, and probabilistic applications.**

***Index Terms*—Memristor, stochastic computing, stochastic number generator, stochastic switching.**

I. Introduction

CONTINUED scaling of CMOS technology to the nanometer scale faces challenges of increasing power dissipation due to leakage and escalating variations [1]. To sustain scaling beyond the CMOS, unconventional device structures and new materials have been proposed with the expectation that they may be able to complement or replace CMOS devices in the future. To incorporate new devices and materials in functional electronic circuits, two common approaches are usually taken: 1) new nanoscale materials or devices used as a channel replacement to improve the mobility of an otherwise conventional transistor geometry, but problems with transistor scaling including power consumption, integration density, and interconnect complexity still remain; 2) nontransistor architectures based on new materials and devices that hold the promise of breaking the barriers of transistor scaling by enabling new computing paradigms are used. A crossbar structure [2]–[5] is one such architecture that is made using two sets of nanowire electrodes that cross each other and form an interconnected network of two-terminal devices (see Fig. 1).

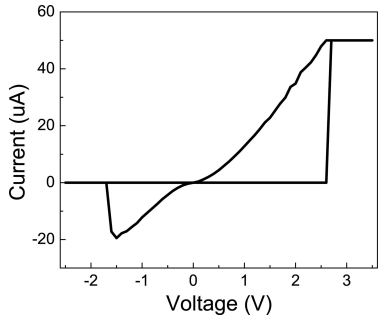

Fig. 1. Current–voltage curve of a digital memristor showing hysteretic resistive switching characteristic with high dynamic range.

A two-terminal device can be made of a pair of top and bottom electrodes and an active material sandwiched in-between. Proper choice of the material can lead to hysteretic resistance switching [6]–[13] as illustrated in Fig. 1. Such a device essentially acts as a nonlinear resistor with memory, and has been termed "memristor" [6], [14], [15].

A. Digital Memristor Device

This study focuses on the use of "digital" memristors as described in [16]. A digital memristor stores binary information, i.e., the low resistance on-state equal to "1" and the high resistance off-state equal to "0," with abrupt resistance changes with on/off ratio of the order of 10^6 as shown in Fig. 1. These digital memristors are "digital" in the sense that they typically have two stable resistance states under given programming conditions, and the switching transition from the high resistance off-state to the low resistance on-state is abrupt.

The high dynamic range of the memristor devices simplifies the read and write operations and improves the robustness. To write a "1" to a memristor, a programming pulse of sufficient duration and voltage $\mathrm{VDD}_{\mathrm{write}}$ is applied to switch the memristor to the ON state. To erase a memristor, i.e., write a "0," a negative $\mathrm{VDD}_{\mathrm{erase}}$ voltage is applied to return the memristor to the OFF state. To read the memristor's value, a reading resistor is connected in series with the $\mathrm{VDD}_{\mathrm{read}}$ supply as shown in Fig. 2. The high-resistance dynamic range allows the memristor

Manuscript received October 15, 2012; accepted January 1, 2014. Date of publication January 16, 2014; date of current version March 6, 2014. This work was supported in part by NSF CCF-1217972. The work of W. Lu was supported by the National Science Foundation CAREER award ECCS-0954621 and in part by the Air Force Office of Scientific Research under MURI grant FA9550-12-1-0038. The review of this paper was arranged by Associate Editor M. R. Stan.

The authors are with the Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor, MI 48109-2122 USA (e-mail: knagphil@umich.edu; wluee@umich.edu; zhengya@eecs.umich.edu).

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

Digital Object Identifier 10.1109/TNANO.2014.2300342

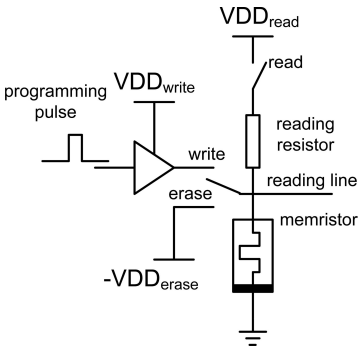

Fig. 2. Read, write, and erase a digital memristor device.

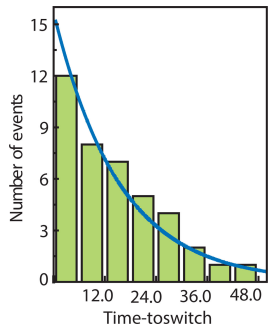

Fig. 3. Histogram of the measured switching time from a single 100-nm memristor device. The blue line is a Poisson fit [20].

values to be read to a nearly full swing digital voltage with a simple resistor divider circuit. Note that $\mathrm{VDD}_{\mathrm{read}}$ is usually much lower than $\mathrm{VDD}_{\mathrm{write}}$ to minimize the possibility of disturbing a memristor's state.

The digital memristors can be built using an M/I/M structure with two conducting electrodes sandwiching a thin insulator in the middle. The abrupt switching characteristic is the result of the formation of a conducting filament that grows when a voltage is applied as shown in Fig. 1. When this filament bridges the gap, the memristor has a low resistance. When a voltage is applied in the opposite direction, the filament will shrink and eventually break, putting the memristor in a high-resistance state.

Recent results have demonstrated functional prototypes of digital memristor devices at feature sizes below 10 nm, switching times below 10 ns, endurance over 10^{12} write/erase cycles, retention time in the order of years, and low programming current under 1 μA, but without the same problems plaguing transistor scaling [9], [12], [13], [17]. Memristor crossbar structures promise key advantages over CMOS transistor circuits in ultrahigh density storage, high-bandwidth connectivity, and convenient reconfiguration. Of particular interest is that memristor devices are CMOS compatible [18]; thus, a memristor-CMOS structure can be built to take advantage of memristor-based high-density storage and routing and efficient CMOS logic circuits. A functional memristor-CMOS prototype has already been demonstrated, consisting of a high-density memristor crossbar vertically integrated on top of CMOS logic circuits, that can be reliably programmed [19].

B. Memristor as an Inherently Stochastic Device

Memristor devices, based on thin metallic-wire electrodes and amorphous or oxide switching layers, are expected to suffer from lower yield and larger variation than conventional devices based on crystalline silicon. Common variation sources include electrode line-edge roughness causing device to device variations, and film thickness irregularity leading to device parameter uncertainty. These spatial variations can be mitigated through variation-aware methods, which has been a well-studied topic in nanometer circuit designs.

Compared to spatial variations, the more challenging problem with memristor devices is the significant randomness from temporal variations. A memristor's resistance switching is stochastic [20]–[23], rather than deterministic as in conventional transistor-based devices. For a "digital" memristor that provides a large dynamic range between logic levels, the change in resistance is associated with the formation and rupture of a dominant, nanoscale conducting filament (either caused by metallic bridge formation [7], [8], [24] or by stoichiometric change in the switching material [25], [26]). Such a resistance switching can now be predicted by physics models, which show that the ion oxidation and transport processes during filament formation are thermodynamically driven and are stochastic in nature for a given filament [7], [20]–[23], [27]. That is, even for the same filament in the same device with the same applied voltage, the switching time is broadly distributed with a statistical average of t_{sw}. This hypothesis has been confirmed by experimental studies that also shown that the switching time follows a Poisson distribution with a characteristic, average time τ (see Fig. 3) [20], [21]. These results all point to the fact that memristors are inherently stochastic devices, and the same operation of the same exact memrsitor device will be accompanied by significant, inherent temporal variations.

Improving memristor's reliability is an active research area, and several approaches have already been proposed: 1) a feedback mechanism to check the output upon every write and adjust the programming voltage and pulse width [28]; 2) error-control coding (ECC) to correct possible errors due to variations [29], [30]; 3) excess programming voltage and long pulse width to guarantee the correctness of each write. Each approach has its own drawback: feedback checking in each write increases the write delay; ECC becomes ineffective when the error rate is high; and the brute-force approach of excess programming voltage and long pulse width costs energy and reduces device lifetime. The extra overhead of the above approaches diminishes the memristor's advantages in density and energy efficiency.

Instead of trying to force the nondeterministic device to operate deterministically, a more promising approach is to design a stochastic computing paradigm to cope with, and even take advantage of, the nondeterminism, which is the rationale behind this study.

C. Stochastic Computing: Preliminaries and Challenges

Stochastic computing was invented in 1967 as a low-cost form of computing based on probabilistic bit streams [31]–[33].

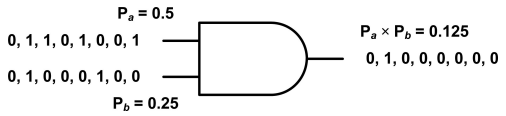

Fig. 4. Stochastic multiplication by a logic AND gate.

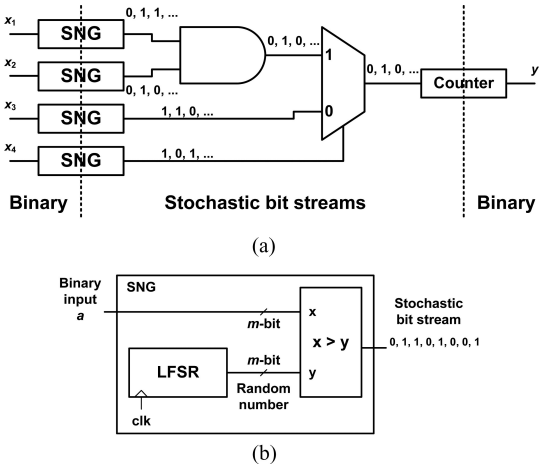

Fig. 5. (a) Stochastic implementation of logic function $y = x_1x_2x_4 + x_3(1 - x_4)$ [34], where SNG and counter are inserted to perform the conversions between binary and stochastic bit streams and (b) LFSR-based implementation of SNG.

For example, the number 0.5 can be represented in stochastic computing by a stream of 8 bits $\{0, 1, 1, 0, 1, 0, 0, 1\}$ such that the probability of finding 1 in a bit is 0.5. In the same way, the number 0.25 can be represented by $\{0, 1, 0, 0, 0, 1, 0, 0\}$. Compared to the common binary numeral system, the probabilistic bit stream representation is not unique, but a longer bit stream provides a higher precision. The bit stream is more error-tolerant than the conventional binary system, as a bit flip introduces an equivalent least significant bit (LSB) error. To use stochastic computing in a binary system, binary numbers are first converted to bit streams and the output of stochastic computing has to be converted to binary.

Stochastic computing fills the niche of low-cost computing, as arithmetic operations can be efficiently implemented. As an example, the multiplication of a and b can be done using an AND logic gate, as shown in Fig. 4. The operation can be understood as follows: by definition of probabilistic bit streams, P_a represents the probability of any bit in stream a being 1; similarly P_b represents the probability of any bit in stream b being 1; and the bitwise AND operation of the two streams produces an output stream, in which the probability of having 1 at a bit position is $P_a \times P_b$, thereby completing the multiplication. The previous calculation assumes that the two input bit streams are independent. Correlation between the streams degrades the accuracy of stochastic computing. For example, if we multiply two identical bit streams represented by a using an AND gate, the product will be P_a, not $P_a \times P_a$.

The independence assumption requires the bit streams to be randomized via stochastic number generator (SNG), as shown in Fig. 5 [34]. The randomization cost presents a significant overhead in stochastic computing, sometimes as high as 80% of the total resource usage [35]. Note that not only the inputs need to be randomized, reshuffling is also necessary at intermediate stages to mitigate the correlations introduced by reconvergent fanouts. The necessity of randomizing bit streams by numerous SNGs partially defeats the simplicity of stochastic computing.

The extra cost of randomization and binary conversion, along with limited precision, have indeed prevented the adoption of stochastic computing. Despite the slow progress, continued research has made the following advances: 1) a large collection of logic, arithmetic, and matrix operations can now be done in stochastic computing [34]–[39], all of which share the elegance of very simple designs and 2) special applications, including artificial neural networks [40]–[42], image processing [35], [43], and decoding of low-density parity-check codes [44], [45] have been successfully demonstrated using stochastic computing. Note the common characteristics among these special applications: 1) error-tolerant and 2) compute-intensive, and the low-cost stochastic computing promises substantial reduction in complexity.

These special applications are of growing importance, as they are closely related to the most rapidly growing application domains including multimedia (image and video), informatics (sensor and social networks), and intelligence (recognition and learning), all of which demand orders of magnitude improvement in compute capability and energy efficiency. High-density, energy-efficient post-CMOS devices such as memristor offer the potential of overcoming the mounting challenges, but the ensuing problem of nondeterministic switching needs to be addressed in a scalable and cost-efficient way.

II. Memristor-Based Native Stochastic Computing

We develop a "native" stochastic computing to exploit the nondeterminism in memristor switching for stochastic computing, as opposed to the conventional attempts to fix the nondeterminism [28]–[30]. The proposed stochastic computing is "native," as the randomness needed in stochastic computing will be intrinsic to the devices and no special addition is needed to generate or ensure randomness. In doing so, we not only obtain the randomness for stochastic computing for free, but also eliminate all the extra energy and complexity required for the deterministic use of memristors. The native stochastic computing based on memristors enables a fundamentally efficient system that is not possible with either memristor or stochastic computing alone.

The envisioned native stochastic computing system is pictured in Fig. 6. The system consists of memristor memories integrated with stochastic arithmetic circuits in a CMOS. The system accepts analog input to be converted to bit stream by a memristor memory. Basic concepts of stochastic bit stream generation have been recently demonstrated experimentally by us [46]. Stochastic computing is performed based on bit streams and the output bit stream is written to memristor memory. Every write to memristor memory allows a new bit stream to be produced (assume that memristor memory is reset before write). The self-contained system described by Fig. 6 is entirely based on bit streams and the binary to bit stream conversions are eliminated. In this way, the native stochastic computing overcomes

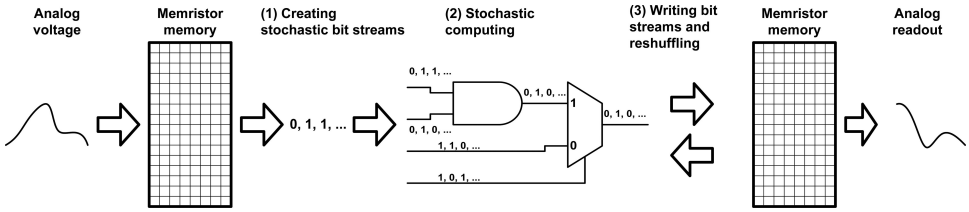

Fig. 6. Native stochastic computing system using memristor-based stochastic memory.

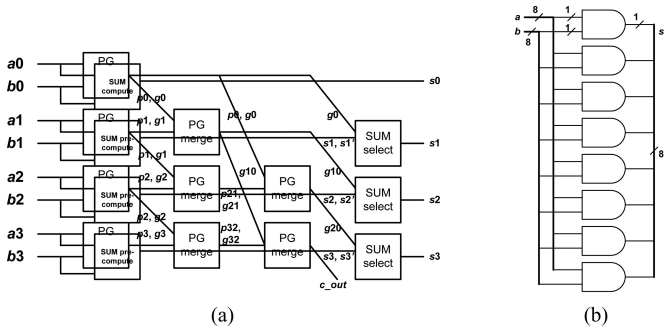

Fig. 7. (a) Binary Kogge–Stone look-ahead adder and (b) parallel stochastic multiplier.

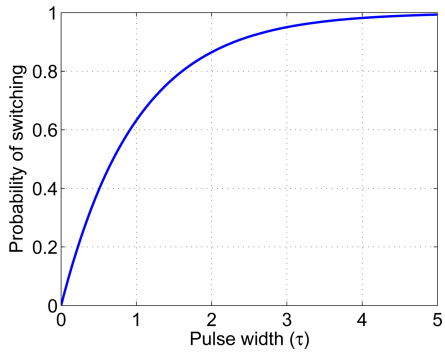

Fig. 8. Memristor switching probability.

two hindrances of classic stochastic computing: 1) the large overhead of stochastic number generation, as randomness does not naturally exist in purely CMOS circuits and must be created algorithmically [35], [47], and 2) the extra conversion steps between binary and bit streams, as the prior designs were never intended to be self-contained systems.

The native stochastic computing system takes advantage of both emerging memristor devices and simple stochastic arithmetic circuits. Since no excess voltage or timing margins are needed to ensure determinism, good energy efficiency can be achieved. Simple stochastic arithmetic circuits can be easily parallelized in a flat topology to deliver high performance. The lack of dependence between bits in a bit stream, in contrast to the bit-level dependence in a binary system, shortens the critical paths and simplifies wiring [an illustration is shown in Fig. 7, where parallelizing a binary adder results in a complex structure and wiring as in Fig. 7(a), compared to a parallel stochastic multiplier that can be efficiently implemented in a flat topology with simple wiring as in Fig. 7(b)]. The native stochastic computing is inherently error-resilient, as the stochastic memory and arithmetic provide tolerance against runtime variations and soft errors.

Note that the native stochastic computing is an end-to-end system that accepts analog inputs directly. Analog inputs may need to be amplified, and a sample and hold is also needed for writing to the memristor. In comparison, the classic stochastic computing is an entirely digital system that requires analog-to-digital conversion to accept analog inputs.

In the following sections, we elaborate on the new technical approaches for each of the three important parts of a native stochastic computing system: 1) creating probablistic bit stream using memristors, 2) writing bit stream to memristors, and 3) carrying out native stochastic computing for practical applications. These three parts are annotated in Fig. 6.

III. Stochastic Programming

A memristor stores 0 in its OFF (high resistance) state and 1 in its ON (low resistance) state. Before programming, the memristor must be reset by applying a negative voltage bias until the memristor enters the high resistance 0 state. To write 1 to a memristor in the 0 state, a positive voltage pulse is applied to turn on the memristor. Energy is consumed in this process, and even after the memristor completes the switching, static current remains on as long as the pulse is ON. It is therefore desirable to turn OFF the pulse whenever the memristor turns ON.

Memristor switching is a stochastic process. Based on prior research, the time to switch follows a Poisson distribution [20]. Given a programming voltage V and pulse width t, the probability of switching is $P(t) = 1 - e^{-t/\tau}$, shown in Fig. 8, where τ is the characteristic switching time that depends on the programming voltage: $\tau(V) = \tau_0 e^{-V/V_0}$ (τ_0 and V_0 are fitting parameters) [20], [21]. For an intuitive idea, if we use a pulse width of $t = \tau$, $P(\tau) = 0.632$, the success rate is too low for a functional memory. If we increase the pulse width to $t = 10\tau$, $P(10\tau) = 0.99995$, the success rate improves but the programming speed is ten times slower and a significant amount of energy is wasted. Alternatively, we can increase the programming voltage V to shorten the necessary pulse width, but it also consumes extra energy and a high voltage accelerates device wearout and shortens its lifetime.

A. Group Write

Instead of trying to ensure a deterministic programming, we opt for an energy-efficient, high-speed stochastic programming using a lower voltage and shorter pulse. Suppose we write 1 to a memristor cell with a pulse width of τ, the success rate is only $P(\tau) = 0.632$. If we apply the pulse to two cells simultaneously, each cell has a 0.632 success rate (assuming each cell switches

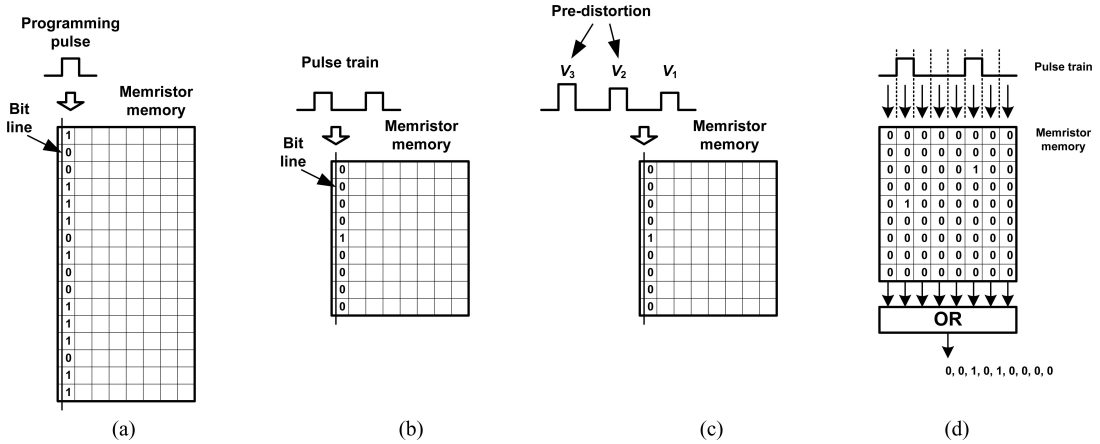

Fig. 9. (a) Writing to a column of memristor cells, (b) stochastic group write to memristor using pulse train, (c) voltage pre-distortion, and (d) parallel single-pulse write.

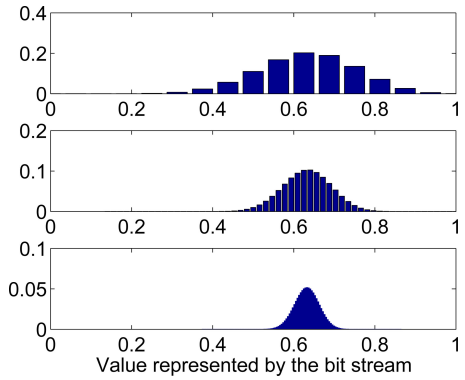

Fig. 10. Distribution of values using an array of 16, 64, and 256 bits (from top to bottom) assuming 0.632 is programmed.

independently) and the expected number of 1's written to the 2 cells is $0.632 \times 2 = 1.264$. If we expand the write to an array of 16 cells, the expected number of 1's is $0.632 \times 16 = 10.112$. In the process of writing to an array of memristor cells, we have essentially accomplished the conversion of the number 0.632 to a stream of 16 bits whose expected number of 1's approximates the given number. We call the write to an array of memristor cells group write. An illustration of group write is shown in Fig. 9(a) and the basic concept was recently demonstrated [46].

Group write reduces the voltage and time required to program memristors which leads to a low-energy consumption. The approach is different from duplication, as write to a larger group of cells yields a higher resolution. For example, group write to 16 cells in Fig. 9(a) produces a 4-bit resolution in a probabilistic fashion. The probabilistic distribution of the stored value depends on the write group size (or bit stream length), as illustrated in Fig. 10. A shorter bit stream sees a larger spread, but it can still be made useful in some practical applications. An added advantage of group write is the resilience against dynamic variations and soft errors, as occasional upsets are unlikely to distort the distribution and cause functional failures.

Group write saves the cost of stochastic number generators (SNG) used in classic stochastic computing. The SNGs are commonly implemented using linear feedback shift register (LFSR) as in Fig. 5(b) [35]. The SNGs generate probabilistic bit streams based on binary inputs, and they are also needed throughout the datapaths to reshuffle bit streams, e.g., at every reconvergent fanout that introduces correlations as one source branches to different paths before reconverging. Reshuffling is done by first converting a bit stream to binary, followed by a SNG to generate a new bit stream. The extensive deployment of SNGs easily overtakes core arithmetic logic as the dominant cost of classic stochastic computing. In comparison, the stochastic programming of an array of memristor cells exploits the randomness native to memristors, thereby eliminating the entire conversion and reshuffling overhead.

Spatial variations in memristors will degrade the accuracy of stochastic number generation by group write. A recent experimental study has showed that the memristor fabrication process can be well controlled, and it also successfully demonstrated stochastic bit stream generation in the space domain [46]. In Section IV, we will further analyze the effects of variation and noise, and demonstrate in Section V the reliable operation through simulations with random voltage noise.

B. Power Estimate

Stochastic programming simplifies stochastic number generation and reduces the power consumption. A 100-MHz SNG made with a 32-bit LFSR and comparator synthesized in a 65-nm CMOS technology is estimated to consume 80.2 μW. The CMOS SNG generates one stochastic bit every clock cycle. The memristor-based stochastic computing generates stochastic bits by simply reading the stochastically programmed memristor values. With a 1 V read supply voltage, a memristor read consumes a static power of 10 μW to read a "1" (i.e., a memristor in the low-resistance state with $R_{\rm on} = 100$ kΩ), and 10 nW to read a "0" (i.e., a memristor in the high-resistance state with $R_{\rm off} = 100$ MΩ). $R_{\rm on}$ and $R_{\rm off}$ are based on fabricated memristor devices. Note that the static power is expected to dominate the total power consumption. With a feedback mechanism, the static current can be turned OFF early; thus, the above power estimates are very conservative. Assuming an equal number of "1" and "0," the average power to generate a stochastic bit using

stochastic programming is approximately 5 μW, a 16$\times$ reduction compared to a CMOS SNG.

The classic CMOS stochastic computing system converts stochastic bits to binary numbers to be stored in memory. The conversion is done using an up counter. A 100 MHz 32-bit up counter synthesized in a 65 nm CMOS technology is estimated to consume 61.4 μW. In a native stochastic computing, the up counter is eliminated and stochastic bits are stored in memristors directly.

The static power for writing a "1" to a memristor is estimated to be 160 μW at a 4 V write supply voltage after the memristor turns ON ($R_{\rm on} = 100$ kΩ). Writing a "0" consumes negligible static power as $R_{\rm off}$ is much higher. Assuming an equal number of "1" and "0," then the average write power is 80 μW. With a feedback mechanism, the static current can be turned OFF early, which will result in a much lower power consumption. Erase power is similar to write power considering the same static current consumption for the respective states except that erase naturally has a cutoff mechanism when the memristors enter the "0" state with a high $R_{\rm off}$ resistance.

The above comparisons demonstrate the potential power efficiency of the memristor-based native stochastic computing over the classic CMOS stochastic computing. We expect the efficiency of using memristors for stochastic computing will continue to improve with improved memristor devices supporting a lower supply voltage and fast feedback mechanisms to limit static current.

C. Erasing Memristors

Erasing memristors to restore the high-resistance state before each write is necessary for the proper operation. Erasing, or resetting, is done by applying a programming voltage of the opposite polarity until the memristor enters the high resistance state. Note that the OFF→ON and ON→OFF switching thresholds are unequal, as shown in Fig. 1, and the characteristic switching times are different. We use OFF→ON switching to stochastically program memristors; and use ON→OFF switching to deterministically erase memristors by adding extra time margin to ensure a correct erase operation. The extra time margin needed to erase increases the latency if the same memristor memory location is continuously being written to. Writing to the same memory location also leads to an uneven wear-out. Therefore, we propose using an erasing scheme similar to what is used in a flash memory, where new data is always written to a fresh memory location and the locations storing stale data are queued to be erased [48]. Erasing will be done on a large block at a time to reduce overhead. This scheme both hides the latency of erasure and ensures an even wear-out by spreading writes evenly to all memory cells.

IV. Compensation of Nonlinear Write to Memristor Memory

In a self-contained stochastic computing system, bit streams are generated from memristor memory for stochastic computing, and the output bit streams of stochastic computing are written to memristor memory. To write a bit stream to memristor memory, we can take one of the two approaches: deterministic or stochastic. In a deterministic write, each bit of the stream is written to one memristor cell in a one-to-one mapping; in a stochastic write, the bit stream is applied to an array of memristor cells using group write. The difference is that the deterministic write produces an exact copy, while a stochastic write reshuffles the bit stream as an elegant way of introducing randomness without the extra reshuffling overhead.

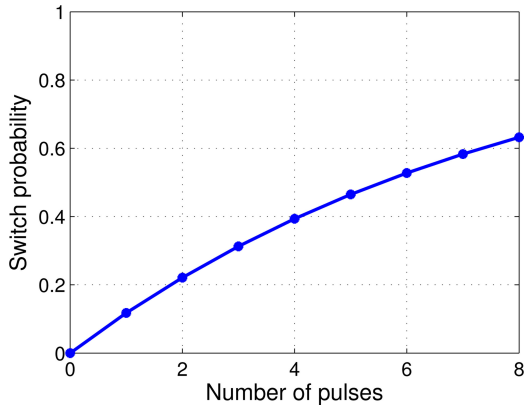

Fig. 11. Probability of switching with number of pulses.

Suppose we apply group write to write a bit stream in the form of pulse train to an array of memristor cells as shown in Fig. 9(b). Assume an 8-bit stream with two 1's (two pulses) to represent 0.25. To preserve the value, we set the pulse voltage for a switching probability of 1/8 = 0.125. After the first pulse is applied to an array of eight memristor cells, we get on average 1 of the 8 cells to switch ON. After the second pulse is applied, the effect of two pulses is experimentally verified to be equivalent to one pulse of twice the width [20]. Based on the model presented in the previous section, the switching probability after each pulse is described in Fig. 11. The relationship between switching probability and number of pulses applied is nonlinear: two pulses give a switching probability of 0.234, slightly below the ideal probability of 0.25. In the extreme case when we apply a train of eight pulses, the switching probability only goes up to 0.656 instead of 1, i.e., only 5.25 of the eight cells will switch ON, resulting in a large error. Therefore, a compensation scheme is needed to undo the nonlinearity.

A. Voltage Predistortion

The nonlinear pulse train write can be compensated using voltage predistortion, illustrated in Fig. 9(c), for an approximation of the ideal linear relationship between switching probability and number of pulses. If a suitably large number of voltage levels are used, voltage predistortion could provide nearly perfect compensation. However, the solution based on numerous voltage levels is expensive. To reduce the cost, we can apply piecewise approximation made from nonlinear functions to reduce the number of voltage levels. A three-piece approximation is shown in Fig. 12 with a relative error limited to 2.5%. Decreasing the error comes at the cost of additional voltage levels, shown in Fig. 13. Compared to a lookup table-based approach,

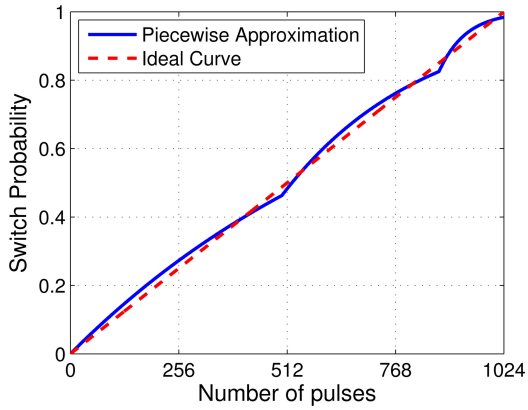

Fig. 12. Piecewise approximation of linear switching probability. The example uses three voltages for less than 2.5% error.

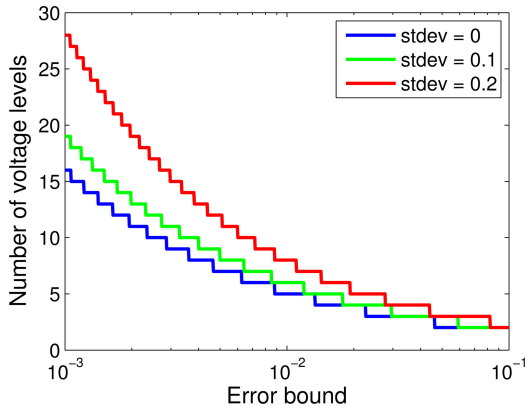

Fig. 13. Number of voltage levels needed to remain under a given error bound using piecewise approximation. Three cases are considered: no voltage noise (stdev = 0), zero-mean Gaussian voltage noise with standard deviation of 0.1V (stdev = 0.1), and zero-mean Gaussian voltage noise with standard deviation of 0.2 V (stdev = 0.2).

the piecewise approximation will be especially handy in long bit streams, while sacrificing only small errors.

Note that voltage predistortion requires a serial write operation, i.e., the pulses have to be applied sequentially. Serializing the write operation presents a potential bottleneck in an inherently parallelizable stochastic computing architecture.

B. Downscaled Write and Upscaled Read

Maintaining numerous voltage levels can be expensive and serial programming slows down the write operation. Futhermore, in the absence of any nonlinear compensation method, the accuracy of pulse train write degrades drastically as the input approaches 1 or full range. This is not surprising since writing a 1 requires the memristor cells to switch with 100% certainty, essentially turning into a deterministic write that is not easily guaranteed in stochastic programming. A downscaled write circumvents this problem by mapping the input to a lower range, e.g., downscaling by a factor of 2 limits the input range from [0, 1] to [0, 0.5]. Within a lower input range, the nonlinearity error becomes much smaller even without compensation. A scalar gain function as described in [20] can be applied in readout, called upscaled read, to undo the downscaled write. The down-

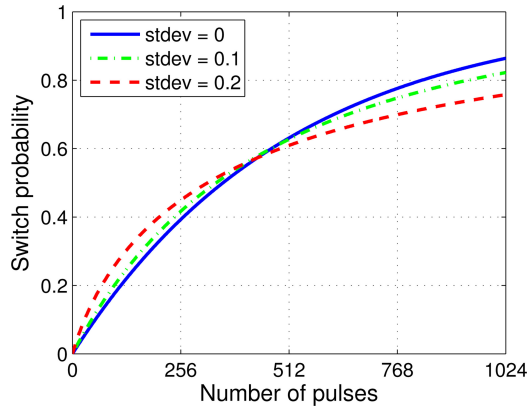

Fig. 14. Memristor switching probability assuming no voltage noise, and zero-mean Gaussian voltage noise of standard deviation = 0.1 and 0.2 V.

scaled write and upscaled read approach uses a single voltage, requires fewer memristors than the parallel single-pulse write, and is also parallelizable. However, this approach degrades the precision due to round-off errors in downscaled mapping.

C. Parallel Single-Pulse Write

Parallel single-pulse write [see Fig. 9(d)] uses a single-pulse voltage in a parallel write. Instead of applying pulses one by one as in voltage predistortion, the entire pulse train will be applied in parallel to a memristor memory. The train is divided into individual pulse segments and each segment is applied to one column of memory. In this way, each column of cells is subject to at most one pulse, thus the name single-pulse. Similar to the downscaled write and upscaled read approach, this scheme takes advantage of the fact that the nonlinear cumulative probability function is relatively linear at the lower end.

The parallel write expands the bit stream representation from a one-dimensional (1-D) array to a 2-D matrix, and an OR function is applied to each row to compress the expanded representation to one single bit stream, as in Fig. 9(d). The given example happens to work perfectly, but a slight problem arises when OR'ing multiple 1's in a row, e.g., OR of two 1's is 1, thus one 1 is lost. The probability of having multiple 1's in a row, or the conflict probability, can be computed beforehand. Based on the conflict probability, the output bit stream can be compensated for a possible loss in value. Alternatively, a stochastic scaled adder followed by a stochastic scalar gain function could be used to correctly read out the stored value. The parallel single-pulse approach has an advantage in terms of implementation cost over the voltage predistortion approach, and it does not suffer from the precision issues of downscaled write and upscaled read, but more memory is used.

D. Variations, Noise, and Calibration

One fundamental difference between the native and the classic stochastic computing is in stochastic number generation. In the classic stochastic computing, stochastic numbers are generated using SNG; whereas in the proposed system, the stochastic numbers are generated by the native stochastic switching of

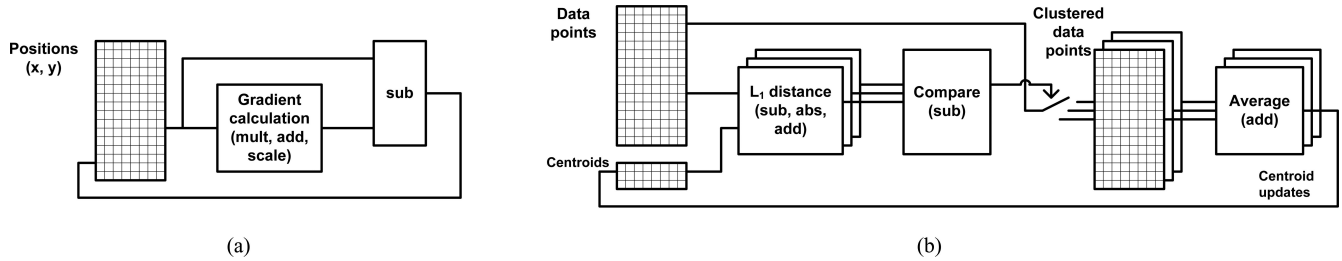

Fig. 15. Stochastic implementation of (a) a gradient descent solver and (b) a k-means clustering processor.

memristors. The memristor switching is affected by variation and noise. In the following, we will analyze the effects of variation and noise, and demonstrate in the next section the reliable operation through simulations with random voltage noise.

The proposed system can be calibrated to accommodate die-to-die process variations and temperature. Process variations manifest themselves in changes of the fit parameters τ_0 and V_0 in the switching probability equation. The effects of die-to-die process variations and temperature can be calibrated out by adjusting the programming voltage, or the width of the programming pulse, or both. Within-die local, device variations can also be calibrated out, but at a higher cost. Therefore, within-die local, variations should be minimized.

Memristor devices on the same die can share close correlations in their device parameters, but note that the correlations in device parameters do not affect the independent switching of each device, i.e., each device will switch independently of the others even though the device parameters are the same or correlated. Independent switching of memristor devices is the basis of the proposed native stochastic computing.

The effect of programming voltage noise can also be calibrated out. Given that the voltage noise v_n follows a defined statistical distribution $f(v_n)$, a memristor's switching probability function is given by

$$P_n = \int_{-\infty}^{\infty} f(v_n)(1 - e^{-\frac{t}{\tau_0 e^{-(V+v_n)/V_0}}})\, dv_n,$$

where $f(v_n)$ is the probability density function of the voltage noise, V is the nominal programming voltage, and τ_0 and V_0 are the fit parameters used in the original switching probability equation. As an example, Fig. 14 shows the memristor switching probability due to Gaussian voltage noise. Random voltage noise changes P_n, but the same nonlinear compensation techniques can be used to fit an updated P_n curve. For example, if voltage predistortion is used, the number of voltage levels needed to remain under a given error bound is given by Fig. 13. Voltage noise will have no effect on the proposed system, as long as the noise distribution is known. Also note that since the switching probability translates into whether a digital memristor is switched ON or OFF, only the mean switching probability P_n is relevant.

Erratic voltage variations, such as occasional voltage droops and oscillations, cannot be calibrated out and they cause inaccuracies in computation. Erratic voltage variations potentially limits the noise floor of stochastic computing. However, the algorithms designed for stochastic computing are often error-tolerant and if such voltage variations happen only intermittently, the system will have a chance to reconverge to the expected accuracy.

V. Applications of Native Stochastic Computing

Native stochastic computing by the integration of memristor memory and stochastic arithmetic circuits offers a new energy-efficient and high-performance computing paradigm. We take advantage of native stochastic computing for data-intensive processing with a soft quality metric—data-intensive so that high-density memristor memory and easily parallelizable stochastic arithmetic circuits can be put to good use, and a soft-quality metric provides the necessary tolerance for a low-cost implementation.

We demonstrate native stochastic computing for two applications: a gradient descent solver and a k-means clustering processor. The results are obtained using three memristor programming techniques: 1) ideal write, 2) voltage predistortion, and 3) downscaled write and upscaled read. We also intentionally add voltage noise to test the robustness of the system.

A. Gradient Descent Solver

Gradient descent is a first-order optimization algorithm used to find the minimum of a cost function [49]. The algorithm repeats two simple steps: 1) calculate the gradient of a given cost function at the current position; 2) move in the negative direction of the gradient by a step proportional to the magnitude of the gradient. If the cost function is well conditioned, the minimum can be obtained by this iterative gradient descent algorithm.

The block diagram of a gradient descent solver is illustrated in Fig. 15(a). The design can be readily translated to a stochastic implementation using memristor memory and stochastic arithmetic circuits. Input positions are stored in memristor memory and the readout is in bit streams. The gradient is calculated using stochastic computing circuits including multiply and add, and the step size is obtained by scalar multiply. The position is updated by the step and stored in memristor memory for the next iteration. Known stochastic designs are available to perform add, multiply, and subtract [31]–[33], [36], [38]. Note that all the arithmetic processing and memory remain in the bit stream domain and no binary conversion is necessary, thus permitting a highly efficient native stochastic computing system.

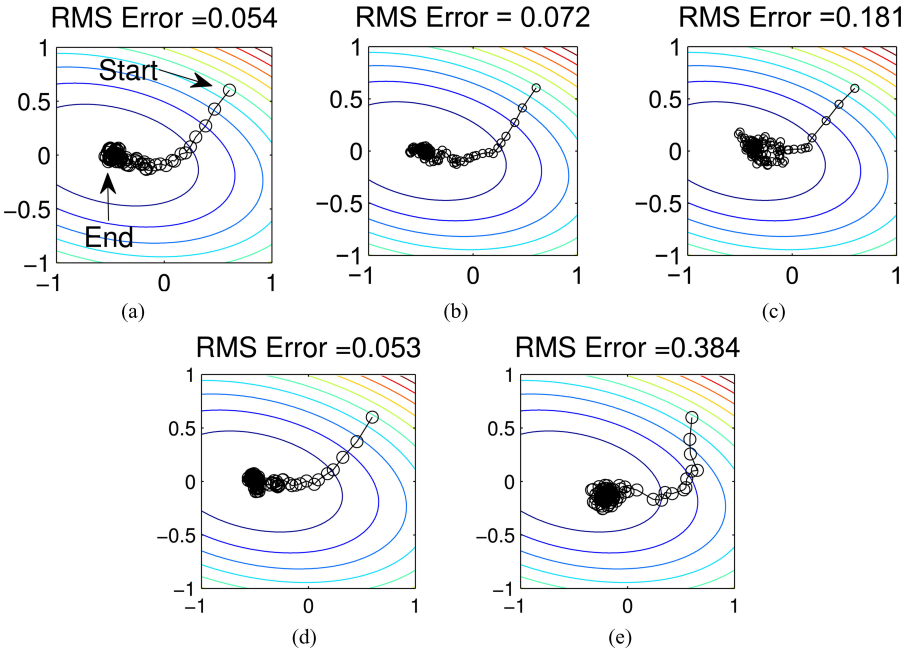

Fig. 16. Stochastic gradient descent algorithm using (a) 32-Kbit stochastic bit stream with ideal write, (b) 32-Kbit stochastic bit stream with voltage predistortion, (c) 256-Kbit stochastic bit stream with downscaled write and upscaled read, (d) 32-Kbit stochastic bit stream with voltage predistortion and zero-mean Gaussian voltage noise of 0.2V standard deviation, and (e) 256-Kbit stochastic bit stream with downscaled write and upscaled read and zero-mean Gaussian voltage noise of 0.2 V standard deviation. The RMS errors from the exact solutions are given for comparison.

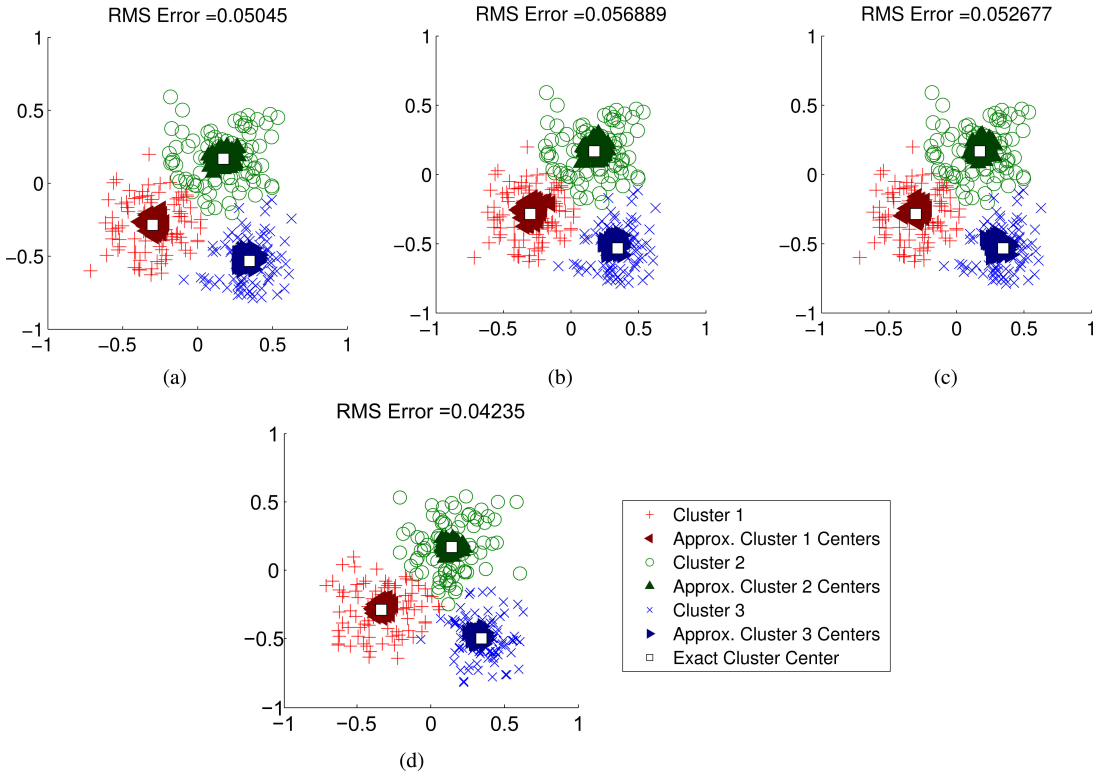

Fig. 17. 256-point k-means clustering with 4-Kbit stochastic bit stream using (a) ideal write, (b) voltage pre-distortion with number of voltage levels chosen to meet 0.1% error bound, (c) voltage pre-distortion with number of voltage levels chosen to meet 0.001% error bound, and (d) voltage predistortion with number of voltage levels chosen to meet 0.1% error bound and zero-mean Gaussian noise of 0.2 V standard deviation. The RMS errors from the exact solutions are given for comparison.

The design is simulated using 32 and 256-Kbit stochastic bit streams to represent bipolar stochastic numbers in the range of $[-1, 1]$. The experiments are based on the cost function of $f(x, y) = \frac{1}{24}((x + 0.5)^2 + (x + 0.5)y + 3y^2)$. Three different memristor programming techniques, ideal write, voltage predistortion, and downscaled write and upscaled read, produce satisfactory results shown in Fig. 16(a)–(c), respectively. Even after voltage noise is added, the computation is shown to be robust as in Fig. 16(d) and (e).

B. k-Means Clustering Processor

In cluster analysis, a set of data points are placed into different clusters whose members are similar, based on a certain metric [50]. Clustering is essential to many applications including image processing, bioinformatics, and machine learning. k-means is a popular clustering algorithm [51] and it is done in three steps: 1) select k cluster centers (centroids); 2) place each data point in one of the clusters to minimize the distance between the data point and the cluster centroid; 3) recompute the centroid of each cluster as the average of all the data points in the cluster. Steps 2) and 3) are repeated until a convergence condition is met.

The block diagram of a k-means processor is illustrated in Fig. 15(b), assuming that $k = 3$ and L_1 distance is used as the similarity metric. Data points and centroids are stored in memristor memory and the readout is in bit streams. The L_1 distance between a data point and each of the centroids is calculated by stochastic subtraction and absolute value operation, the results of which are compared using stochastic subtraction and comparison. The data point is written to the respective cluster memory based on the shortest L_1 distance. Once a round of clustering is done, stochastic averaging is carried out to update the cluster centroids.

Examples of the k-means clustering using stochastic computing and memristor programing techniques are simulated using 4-Kbit stochastic bit streams to represent bipolar stochastic numbers in the range of $[-1, 1]$. 256-point datasets are placed in three clusters such that the L_1 distance is minimized to the cluster centroids. The two different memristor programming techniques, ideal write and voltage-predistortion, produce satisfactory results shown in Fig. 17. The computation is robust against voltage noise, as seen in Fig. 17(d).

VI. Conclusion

Two-terminal memristor devices are inherently stochastic devices that require extra energy and latency to enforce deterministic behavior. This study takes advantage of the memristor's stochastic behavior to produce random bit streams needed in stochastic computing. In the proposed approach, memristors replace stochastic number generators in a native stochastic computing architecture.

We present group write to program the memristor memory cells in arrays to generate the random bit streams for stochastic computing. To enable linear write to memristor memory, we propose compensation techniques including voltage predistortion, downscaled write and upscaled read, and parallel single-pulse write. We evaluate the native stochastic computing architecture by simulating a gradient descent solver and a k-means clustering processor. Group write together with nonlinearity compensation techniques are shown to be effective for stochastic memristor programming. The proposed native stochastic computing architecture takes advantage of the key benefits of both stochastic computing and memristor devices to enable a new low-energy, high-performance, and low-cost computing paradigm.

Acknowledgment

The authors would like to thank S. Gaba for memristor measurement data and helpful discussions.

References

[1] (2010). "International technology roadmap for semiconductors. 2010 update," [Online]. Available: http://public.itrs.net/

[2] W. Lu and C. Lieber, "Nanoelectronics from the bottom up," *Nat. Mater.*, vol. 6, no. 11, pp. 841–850, 2007.

[3] K. Likharev and D. Strukov, "CMOL: devices, circuits, and architectures," *Introducing Mol. Electron.*, vol. 680, pp. 447–477, 2005.

[4] G. Snider, "Computing with hysteretic resistor crossbars," *Appl. Phys. A, Mater. Sci. Process.*, vol. 80, no. 6, pp. 1165–1172, 2005.

[5] P. Kuekes, D. Stewart, and R. Williams, "The crossbar latch: Logic value storage, restoration, and inversion in crossbar circuits," *J. Appl. Phys.*, vol. 97, no. 3, pp. 034 301.1–034 301.5, 2005.

[6] D. Strukov, G. Snider, D. Stewart, and R. Williams, "The missing memristor found," *Nature*, vol. 453, no. 7191, pp. 80–83, 2008.

[7] R. Waser and M. Aono, "Nanoionics-based resistive switching memories," *Nat. Mater.*, vol. 6, no. 11, pp. 833–840, 2007.

[8] R. Waser, R. Dittmann, G. Staikov, and K. Szot, "Redox-based resistive switching memories–nanoionic mechanisms, prospects, and challenges," *Adv. Mater.*, vol. 21, nos. 25–26, pp. 2632–2663, 2009.

[9] M. Kozicki, M. Park, and M. Mitkova, "Nanoscale memory elements based on solid-state electrolytes," *IEEE Trans. Nanotechnol.*, vol. 4, no. 3, pp. 331–338, May 2005.

[10] I. Valov, R. Waser, J. Jameson, and M. Kozicki, "Electrochemical metallization memories: Fundamentals, applications, prospects," *Nanotechnology*, vol. 22, no. 25, pp. 1–22, 2011.

[11] C. Cheng, C. Tsai, A. Chin, and F. Yeh, "High performance ultra-low energy RRAM with good retention and endurance," in *Proc. IEEE Int. Electron Devices Meet.*, 2010, pp. 19.4.1–19.4.4.

[12] B. Govoreanu, G. Kar, Y. Chen, V. Paraschiv, S. Kubicek, A. Fantini, I. P. Radu, L. Goux, S. Clima, R. Degraeve, N. Jossart, O. Richard, T. Vandeweyer, K. Seo, P. Hendrickx, G. Pourtois, H. Bender, L. Altimime, D. Wouters, J. Kittl, and M. Jurczak, "10 $\times$ 10 nm2 Hf/HfOx crossbar resistive RAM with excellent performance, reliability and low-energy operation," in *Proc. IEEE Int. Electron Devices Meet.*, 2011, pp. 31.6.1–31.6.4.

[13] M. Lee, C. Lee, D. Lee, S. Lee, M. Chang, J. Hur, Y. Kim, C. Kim, D. Seo, S. Seo, U. Chung, I. Yoo, and K. Kim, "A fast, high-endurance and scalable non-volatile memory device made from asymmetric Ta2O5-x/TaO2-x bilayer structures," *Nat. Mater.*, vol. 10, no. 8, pp. 625–630, 2011.

[14] L. Chua, "Memristor-the missing circuit element," *IEEE Trans. Circuit Theory*, vol. CT-18, no. 5, pp. 507–519, Sep. 1971.

[15] L. Chua and S. Kang, "Memristive devices and systems," *Proc. IEEE*, vol. 64, no. 2, pp. 209–223, Feb. 1976.

[16] W. Lu, K.-H. Kim, T. Chang, and S. Gaba, "Two-terminal resistive switches (memristors) for memory and logic applications," in *Proc. Asia South Pacif. Design Autom. Conf.*, 2011, pp. 217–223.

[17] K. Kim, S. Jo, S. Gaba, and W. Lu, "Nanoscale resistive memory with intrinsic diode characteristics and long endurance," *Appl. Phys. Lett.*, vol. 96, no. 5, pp. 053 106.1–053 106.3, 2010.

[18] S. Jo and W. Lu, "CMOS compatible nanoscale nonvolatile resistance switching memory," *Nano Lett.*, vol. 8, no. 2, pp. 392–397, 2008.

[19] K. Kim, S. Gaba, D. Wheeler, J. Cruz-Albrecht, T. Hussain, and N. Srinivasa, W. Lu, "A functional hybrid memristor

crossbar-array/CMOS system for data storage and neuromorphic applications," *Nano Lett.*, vol. 12, no. 1, 2012.
[20] S. Jo, K. Kim, and W. Lu, "Programmable resistance switching in nanoscale two-terminal devices," *Nano Lett.*, vol. 9, no. 1, pp. 496–500, 2008.
[21] D. Strukov, J. Borghetti, and R. Williams, "Coupled ionic and electronic transport model of thin-film semiconductor memristive behavior," *Small*, vol. 5, no. 9, pp. 1058–1063, 2009.
[22] S. Savelev, A. Alexandrov, A. Bratkovsky, and R. Williams, "Molecular dynamics simulations of oxide memristors: Thermal effects," *Appl. Phys. A, Mater. Sci. Process.*, vol. 102, no. 4, pp. 891–895, 2011.
[23] X. Ma and K. Likharev, "Global reinforcement learning in neural networks with stochastic synapses," in *Proc. Int. Joint Conf. Neural Netw.*, 2006, pp. 47–53.
[24] Y. Yang, P. Gao, S. Gaba, T. Chang, X. Pan, and W. Lu, "Observation of conducting filament growth in nanoscale resistive memories," *Nat. Commun.*, vol. 3, no. 732, pp. 1–8, 2012.
[25] D. Kwon, K. Kim, J. Jang, J. Jeon, M. Lee, G. Kim, X. Li, G. Park, B. Lee, S. Han, M. Kim, and C. Hwang, "Atomic structure of conducting nanofilaments in TiO2 resistive switching memory," *Nat. Nanotechnol.*, vol. 5, no. 2, pp. 148–153, 2010.
[26] J. Strachan, M. Pickett, J. Yang, S. Aloni, A. D. Kilcoyne, G. Medeiros-Ribeiro, and R. Williams, "Direct identification of the conducting channels in a functioning memristive device," *Adv. Mater.*, vol. 22, no. 32, pp. 3573–3577, 2010.
[27] P. Sheridan, K. Kim, S. Gaba, T. Chang, L. Chen, and W. Lu, "Device and SPICE modeling of RRAM devices," *Nanoscale*, vol. 3, no. 9, pp. 3833–3840, 2011.
[28] K. Jo, C. Jung, K. Min, and S. Kang, "Self-adaptive write circuit for low-power and variation-tolerant memristors," *IEEE Trans. Nanotechnol.*, vol. 9, no. 6, pp. 675–678, Nov. 2010.
[29] P. Kuekes, W. Robinett, R. Roth, G. Seroussi, G. Snider, and R. Williams, "Resistor-logic demultiplexers for nanoelectronics based on constant-weight codes," *Nanotechnology*, vol. 17, no. 4, pp. 1052–1061, 2006.
[30] P. Kuekes, W. Robinett, and R. Williams, "Improved voltage margins using linear error-correcting codes in resistor-logic demultiplexers for nanoelectronics," *Nanotechnology*, vol. 16, no. 9, pp. 1419–1432, 2005.
[31] B. Gaines, "Stochastic computing," in *Proc. Spring Joint Comput. Conf.*, 1967, pp. 149–156.
[32] W. Poppelbaum, C. Afuso, and J. Esch, "Stochastic computing elements and systems," in *Proc. Fall Joint Comput. Conf.*, 1967, pp. 635–644.
[33] S. Ribeiro, "Random-pulse machines," *IEEE Trans. Electron. Comput.*, vol. EC-16, no. 3, pp. 261–276, Jun. 1967.
[34] X. Li, W. Qian, M. Riedel, K. Bazargan, and D. Lilja, "A reconfigurable stochastic architecture for highly reliable computing," in *Proc. Great Lakes Symp. VLSI*, 2009, pp. 315–320.
[35] W. Qian, X. Li, M. Riedel, K. Bazargan, and D. Lilja, "An architecture for fault-tolerant computation with stochastic logic," *IEEE Trans. Comput.*, vol. 60, no. 1, pp. 93–105, Jan. 2011.
[36] W. Qian and M. Riedel, "The synthesis of robust polynomial arithmetic with stochastic logic," in *Proc. Design Autom. Conf.*, 2008, pp. 648–653.
[37] P. Mars and H. Mclean, "High-speed matrix inversion by stochastic computer," *Electron. Lett.*, vol. 12, no. 18, pp. 457–459, 1976.
[38] S. Toral, J. Quero, and L. Franquelo, "Stochastic pulse coded arithmetic," in *Proc. IEEE Int. Symp. Circuits Syst.*, 2000, vol. 1, pp. 599–602.
[39] J. Keane and L. Atlas, "Impulses and stochastic arithmetic for signal processing," in *Proc. IEEE Int. Conf. Acoust., Speech, Signal Process.*, 2001, vol. 2, pp. 1257–1260.
[40] J. Dickson, R. McLeod, and H. Card, "Stochastic arithmetic implementations of neural networks with in situ learning," in *Proc. IEEE Int. Conf. Neural Netw.*, 1993, pp. 711–716.
[41] Y. Kim and M. Shanblatt, "Architecture and statistical model of a pulse-mode digital multilayer neural network," *IEEE Trans. Neural Netw.*, vol. 6, no. 5, pp. 1109–1118, Sep. 1995.
[42] B. Brown and H. Card, "Stochastic neural computation—Part I: Computational elements," *IEEE Trans. Comput.*, vol. 50, no. 9, pp. 891–905, Sep. 2001.
[43] T. Hammadou, M. Nilson, A. Bermak, and P. Ogunbona, "A 96×64 intelligent digital pixel array with extended binary stochastic arithmetic," in *Proc. Int. Symp. Circuits Syst.*, 2003, vol. 4, pp. IV-772–IV-775.
[44] V. Gaudet and A. Rapley, "Iterative decoding using stochastic computation," *Electron. Lett.*, vol. 39, no. 3, pp. 299–301, 2003.
[45] S. Sharifi Tehrani, W. Gross, and S. Mannor, "Stochastic decoding of LDPC codes," *IEEE Commun. Lett.*, vol. 10, no. 10, pp. 716–718, Oct. 2006.
[46] S. Gaba, P. Sheridan, J. Zhou, S. Choi, and W. Lu, "Stochastic memristive devices for computing and neuromorphic applications," *Nanoscale*, vol. 5, pp. 5872–5878, 2013.
[47] P. Jeavons, D. Cohen, and J. Shawe-Taylor, "Generating binary sequences for stochastic computing," *IEEE Trans. Inf. Theory*, vol. 40, no. 3, pp. 716–720, May 1994.
[48] E. Gal and S. Toledo, "Mapping structures for flash memories: Techniques and open problems," in *Proc. IEEE Int. Conf. Softw., Sci., Technol. Eng.*, 2005, pp. 83–92.
[49] J. Nocedal and S. J. Wright, *Numerical Optimization*, 2nd ed. New York, NY, USA: Springer-Verlag, 2006.
[50] L. Kaufman and P. Rousseeuw, *Finding Groups in Data: An Introduction to Cluster Analysis*. New York, NY, USA: Wiley Online Library, 1990.
[51] J. MacQueen, "Some methods for classification and analysis of multivariate observations," in *Proc. Berkeley Symp. Math. Statist. Probabil.*, 1967, vol. 1, pp. 281–297.

Phil Knag (S'11) received the B.S. degree in computer engineering and the M.S. degree in electrical engineering from the University of Michigan, Ann Arbor, MI, USA, in 2010 and 2012, respectively, where he is currently working toward the Ph.D. degree in electrical engineering.

He received a GAANN fellowship in 2010 from the U.S. Department of Education for academic excellence. He was with Medtronic, Inc. as a Research-Intern in 2010. His current research interests include nanoscale and neuromorphic computing systems.

Wei Lu (M'05) received the B.S. degree in physics from Tsinghua University, Beijing, China, in 1996, and the Ph.D. degree in physics from Rice University, Houston, TX, USA, in 2003.

From 2003 to 2005, he was a Postdoctoral Research Fellow at Harvard University, Cambridge, MA, USA. In 2005, he joined the faculty of the Department of Electrical Engineering and Computer Science, the University of Michigan, Ann Arbor, MI, USA, and is currently an Associate Professor. His research interests includes high-density memory based on two-terminal resistive switches (RRAM), memristor-based logic circuits, aggressively scaled transistor devices, and electrical transport in low-dimensional systems.

Dr. Lu is an Editor-in-Chief for *Nanoscale*, a member of the IEEE, APS, MRS, an active member of several IEEE technical committees and program committees. He has received the NSF CAREER Award.

Zhengya Zhang (S'02–M'09) received the B.A.Sc. degree in computer engineering from the University of Waterloo, Waterloo, ON, Canada, in 2003, and the M.S. and Ph.D. degrees in electrical engineering from the University of California, Berkeley, CA, USA, in 2005 and 2009, respectively.

Since 2009, he has been with the faculty of the University of Michigan, Ann Arbor, MI, USA, as an Assistant Professor in the Department of Electrical Engineering and Computer Science. His current research interests include low-power and high-performance VLSI circuits and systems for computing communications and signal processing.

Dr. Zhang received the National Science Foundation CAREER Award in 2011, the Intel Early Career Faculty Honor Program Award in 2013, the David J. Sakrison Memorial Prize for outstanding doctoral research in electrical engineering and computer Science at UC Berkeley, and the Best Student Paper Award at the Symposium on VLSI Circuits. He is an Associate Editor of the IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS–I, REGULAR PAPERS.