

Efficient Memristive Stochastic Differential Equation Solver

Xuening Dong, Louis Primeau, Roman Genov, Mostafa Rahimi Azghadi,*
and Amirali Amirsoleimani

Herein, an efficient numerical solver for stochastic differential equations based on memristors is presented. The solver utilizes the stochastic switching effect in memristive devices to simulate the generation of a Brownian path and employs iterative Euler method computations within memristive crossbars. The correctness of the solution paths generated by the system is examined by solving the Black–Scholes equations and comparing the paths to analytical solutions. It is found that the absolute error of a 128-step path is limited to an order of 10^{-2} . The tolerance of the system to crossbar nonidealities is also assessed by comparing the numerical and analytical paths' variation in error. The numerical solver is sensitive to the variation in operating conditions, with the error increasing by $1.17\times$, $38.7\times$, and $1222\times$ as the ambient temperature, wire resistance, and stuck probability of the memristor increase to extreme conditions. The solver is tested on a variety of problems to show its utility for different calculations. And, the resource consumption of the proposed structure built with existing technology is estimated and it is compared with similar iterative solvers. The solver generates a solution with the same level of accuracy from $4\times$ to $10\times$ faster than similar digital or mixed-signal designs.

dynamical systems with unknown and fluctuating natures. Furthermore, recent studies have delved into the use of SDEs in generative modeling for injecting/removing noise from complex data.^[2] For complex models, analytical solutions are rare; instead, numerical methods such as Monte Carlo simulation are widely used for solving SDE systems in which multiple realizations of the same equation are repeated with different samples from the noisy process. Each realization is given an equal probability of forming the final solution, yielding a probability distribution representing the possible outcomes of the solution.

The two key elements of Monte Carlo simulation for stochastic differential equation (SDEs) are the computation of the deterministic functions and forward timestep and the simulation of the Brownian motion path for the stochastic term. In this work, for the deterministic

1. Introduction

Stochastic differential equation (SDE) theory has been introduced to various fields of research, including finance, biological modeling, and mechanics.^[1] The inclusion of stochastic terms in differential equations provides a useful way to modeling

term, we focus on the Euler–Maruyama method, which is one of the simplest numerical methods for solving SDEs based on the Ito–Taylor expansion.^[1] The 1D Brownian motion paths, also known as the Wiener processes,^[3] are generated by continuous Gaussian random numbers (GRNs).


Software implementations of these methods, such as provided in refs. [4,5], iterate through steps on the solution paths and simulate stochastic processes with pseudo-random software-based random number generators (RNGs). As the step size decreases to attain better accuracy, the number of iterations increases, demanding more access to memory for retrieving data from previous iterations (Figure 1a). This process leads to high energy demand and long processing time due to serial data processing in the Von Neumann architecture,^[6–8] making the software implementations power and time hungry. To address this problem, we propose to use the well-known concept of in-memory computing, as shown in Figure 1b for solving SDEs.

Memristors with their nonlinear electrical behavior and nonvolatile characteristics have been widely configured as memory elements, logic gates, or computational devices. This device has an adjustable resistance that not only stores data but also modulates incoming data flow to perform logic or arithmetic operations. Organizing such devices in a crossbar manner helps to deal with a large amount of data in parallel and perform ultraefficient vector–matrix multiplication (VMM) based on the fundamental Ohm and Kirchhoff laws.^[9,10] The numerical solving process of SDEs can thereby be adapted to the system

X. Dong, L. Primeau, R. Genov
Edward S. Rogers Sr. Department of Electrical and Computer Engineering
University of Toronto
10 King's College Road, Toronto M5R 0A3, Ontario, Canada

M. Rahimi Azghadi
College of Science and Engineering
James Cook University
Townsville, QLD 4811, Australia
E-mail: mostafa.rahimiazghadi@jcu.edu.au

A. Amirsoleimani
Lassonde School of Engineering
York University
Toronto M3J 1P3, Ontario, Canada

 The ORCID identification number(s) for the author(s) of this article can be found under <https://doi.org/10.1002/aisy.202300008>.

© 2023 The Authors. Advanced Intelligent Systems published by Wiley-VCH GmbH. This is an open access article under the terms of the Creative Commons Attribution License, which permits use, distribution and reproduction in any medium, provided the original work is properly cited.

DOI: 10.1002/aisy.202300008

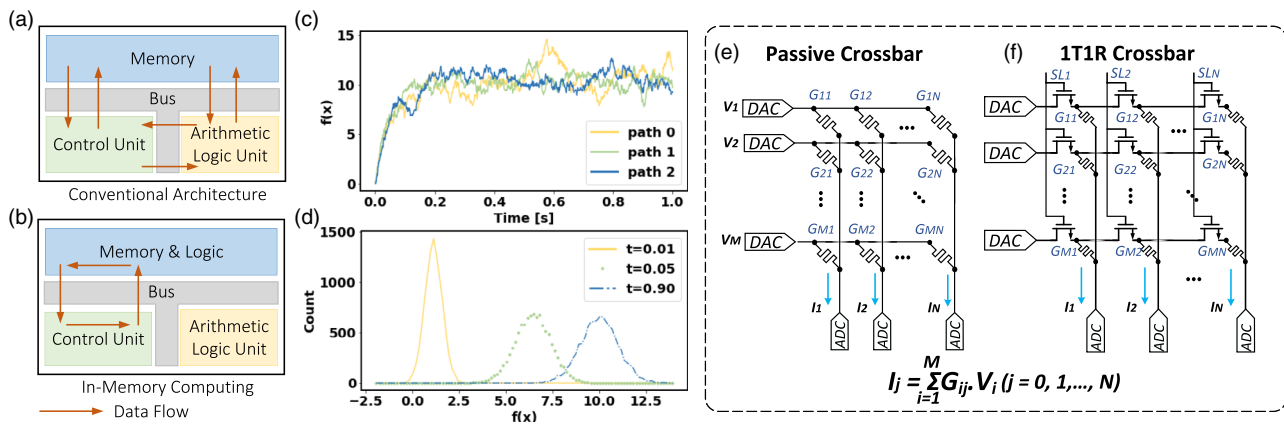


Figure 1. The dataflow within a) conventional computer architecture and b) in-memory computing systems. c) Three solution paths to the Ornstein-Uhlenbeck process were constructed from the Euler-Maruyama method using three different stochastic processes. d) The time evolution of the distribution of the stochastic processes. The distribution of solution values at three timestamps (0.01, 0.05, and 0.9 s) from 10 000 paths is plotted. The circuit diagram of e) passive and f) one-transistor-one-memristor (1T1R) crossbars that are considered for vector-matrix multiplication (VMM) in hardware computing systems.

by transforming the iterative steps into matrix and vector form using the Euler-Maruyama method.

Despite the efficiency they exhibit in calculations, the roadblock to developing memristive devices at large scales for accurate calculations is the variation in their switching parameters.^[11] The inherent switching parameters vary from device to device and even cycle to cycle in the same device, causing spatial and temporal randomness introduced into every operation.^[12] This randomness can be harnessed for its true stochastic behavior, and used for stochastic computing,^[12–14] stochastic neural networks,^[11,15] or RNG.^[16,17]

The nonidealities of the entire memristor crossbar system as a whole are another concern to the accuracy of the calculation. Issues such as sneak-path current^[18] and wire resistance^[19] can alter the output current flow and lead to error in the final solution. Therefore, accuracy enhancement techniques are necessary to minimize the impact these nonidealities have on the computations.^[20]

In this work, we propose a memristor-based SDE solver structure that is based on the two aforementioned properties of memristors and memristor crossbars, that is, suitability of in-memory computing, and intrinsic stochastic behavior. Our specific contributions are as follows: 1) We introduce the use of memristor-based Gaussian random number generator (GRNG) to generate GRNs iteratively based on crossbar VMM to simulate the Brownian motion path. The RNG is built on the intrinsic stochastic switching behavior of memristors to enhance the quality of randomness in the Gaussian path generations. 2) We implement an iterative solver for realizing a solution path to SDEs using Euler-Maruyama method by performing all the multiplication and accumulation steps in-memory. 3) We enhance the accuracy of the system with bit-slicing methods to cut down the effect of nonidealities in the crossbar. 4) We propose a process-slicing method that reduces the crossbar size needed for solution paths of more than 100 steps to minimize the simulation time while maintaining the accuracy of the solutions. 5) We apply the proposed structure to real-world problems and

compare the quality of simulated results with software numerical solutions.

The rest of the paper is structured as follows: in Section 2, we introduce the concept of SDEs and their numerical solving methods. In Section 3, we overview the scientific computing hardware. In Section 4, we present the proposed solver structure and associated precision enhancement techniques. In Section 5, we report and discuss simulation results under nonideal conditions and in Section 6, we overview and discuss the results from real-world problems. In Section 7, we analyze the resource consumption of the solver. Finally, we conclude the work in Section 8.

2. Stochastic Differential Equations

An SDE is a type of differential equation with random coefficients, either constants or functions, to introduce perturbations to the deterministic term.^[21] Common SDEs are seen in the following form

$$dX = \underbrace{\mu(t, X(t))dt}_{\text{deterministic term}} + \underbrace{\sigma(t, X(t))dW(t)}_{\text{stochastic term}} \quad (1)$$

where μ and σ are given functions, and $dW(t)$ is a Brownian motion. The Brownian motion is defined as a path that starts from origin and has continuous independent increments sampled from a normal distribution with zero mean and variance equaling the step size.^[4]

SDEs can be considered in the Ito sense or the Stratonovich sense, depending on the discretization of the stochastic process in the integral. In this work, we focus on the 1D Ito sense SDE and propose a solver derived from Ito-Taylor expansion.^[1]

2.1. Numerical Methods for SDEs

Numerical methods for SDEs can be broadly divided into two classes: those that simulate many stochastic trajectories, which

is the way analytic solutions (if they exist) are generated, and those that solve the corresponding Fokker–Planck partial differential Equation (2), which we focus on in this paper.

$$\frac{\partial p}{\partial t} = -\frac{\partial}{\partial x}(rp) + \frac{1}{2}\frac{\partial^2}{\partial x^2}(\sigma^2 p) \quad (2)$$

where p is the probability density function of random variable X , x are the values that random variable X can take and r and σ are the deterministic and stochastic functions, respectively. The simplest ordinary differential equation (ODE) solver for SDEs is the Euler–Maruyama method (Figure 1c,d), which has the following form

$$X_n - X_{n+1} = \mu(t_n, X_n)\Delta t + \sigma(t_n, X_n)\Delta W \quad (3)$$

where X_n is the value of the process at the n th step. Simulation of the SDE involves iterating this equation for a desired amount of time steps, where the size of the time steps dictates the accuracy of the solution. The Euler–Maruyama method aforementioned converges strongly with order $1/2$, which means that it has the following relation between the error and the step size

$$E[X(T) - w_{\Delta t}(T)] = O((\Delta t)^{\frac{1}{2}}) \quad (4)$$

where $X(T)$ is the continuous stochastic process (true solution) and $w(\Delta t)$ is the discrete approximation that we obtain using a differential equation solver. As we reduce the time step size Δt , the error decreases. We can also consider convergence in the weak sense, which has the definition that

$$E[f(X(T))] - E[f(w_{\Delta t}(T))] = O(\Delta t) \quad (5)$$

for all polynomials f of the stochastic process, continuous or approximate. This means that the distribution of the trajectories is guaranteed to converge in the limit of small time steps. Note that the Euler–Maruyama converges with order 1 in the weak sense. Because of the relaxed definition, it is possible to get solvers with higher weak convergence orders for less computational complexity.

2.2. Applications

Taking advantage of the instantaneous noise modeled by the stochastic term, SDEs are suitable for modeling dynamic systems that evolve randomly with time.^[22] They generalize real-world random phenomena including financial pricing with volatility,^[23,24] stochastic thermodynamics,^[25] and biological modeling of stochastic voltage-gated channel operation in the deterministic Hodgkin–Huxley model.^[26] Recent work in machine-learning frameworks also consider stochastic noise injection modeled by neural SDEs.^[22,27,28] The randomness of the model covers the Gaussian noise injection, dropout, and other perspectives of the neural network,^[27] and can be solved by fitting the SDE into Wasserstein generative adversarial network.^[28]

3. Scientific Computing Hardware and Methods

The two core operations in solving an SDE are the generation of a random Brownian path and the computation of matrix–vector

multiplication, which can be efficiently implemented using an in-memory architecture. For this architecture, a nonvolatile memory is needed for both recording the data and performing arithmetic operations, in-memory.

3.1. Memristor Crossbars

Based on the programmable resistance nature, memristors provide a promising way for performing analogue computations by directly modulating the current passing through them. By arranging such a device in a crossbar structure, with a word line representing the input voltage and the output currents summed up in the bit line, the whole system can perform the “multiply-and-accumulate” operation and is thus used for VMMs.

Within the crossbar structure, all the calculations can be performed in parallel simultaneously.^[7] This reduces the time and power used to perform computations. Commonly seen memristor crossbars include a gateless passive (1R) structure (Figure 1e) or a one-transistor-one-memristor (1T1R) structure (Figure 1f). The memristor-only structure has lower power consumption but suffers from sneak-path current problems. The 1T1R structure resolves the sneak-path problem by controlling the transistor states, with only leakage current when a gate is turned OFF. Both structures suffer from device and crossbar nonidealities such as ageing issues, wire resistance, and imprecise writing of memristor conductance, which limit the precision of calculations. To date, the focus of memristive crossbar computing systems has been mostly on tasks such as inference and training of artificial neural networks,^[29–31] which usually have a high tolerance to errors. Numerical computation tasks which have high-precision requisites are challenging to implement on such systems. Therefore, to compensate for the loss, a series of precision–extension steps have been developed to be carried out during the numerical calculations using memristive crossbars.^[20]

3.2. Stochastic Switching Behavior

A memristor’s conductance is dependent on the previous amount of electric charge that has flown through it.^[32] Therefore, the conductance or resistance is nondeterministic and varies with the voltage pulse applied (Figure 2a). The conductance is also bounded by a set of minimum and maximum values, known as the ON and OFF resistance, following a normal distribution at the device level (Figure 2b,c). The outputs of the high-resistance state (HRS) and low-resistance state (LRS) are then defined as logic 0 and 1. To perform a SET operation, a positive input voltage is applied to one terminal until an abrupt jump in the current flows out of the other terminal. The RESET operation can be performed in the reversed direction or by applying a negative input, switching the memristor from the ON to the OFF state. The actual switching between the two states is nondeterministic, depending on the structure and doping material from which the memristor has been manufactured. For example, in filament-based devices, the formation or dissolution of conductive filament between the top and bottom electrodes is completely random and device independent. Therefore, the actual resistance and the resultant current at LRS and HRS vary from device to

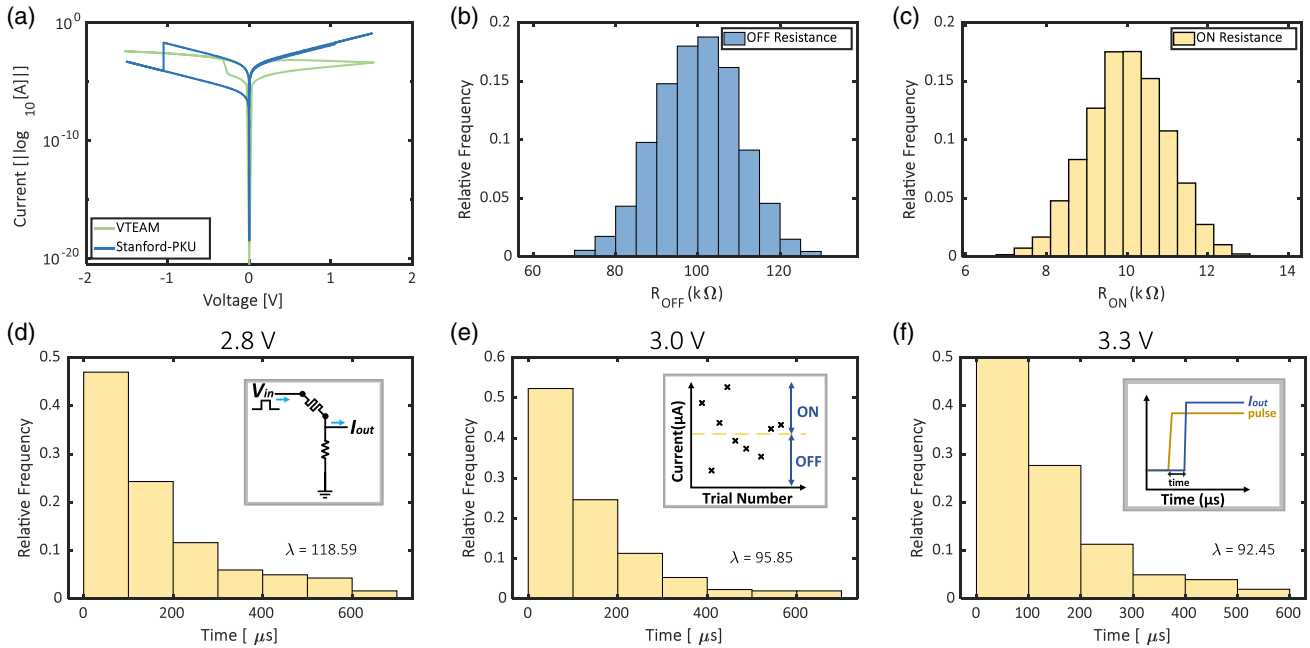


Figure 2. a) The bipolar switching behavior of two memristor models, VTEAM and Stanford-PKU, simulated on the MemTorch platform^[29] within the voltage range of ± 1.5 V. The memristors used for crossbar simulation are set with b) OFF and c) ON resistance in a crossbar following a normal distribution. The distribution of the switching time simulated on a stochastic memristor model following a Poisson distribution. The switching time is defined as the difference between the onset of the applied pulse and the jump of output current from the memristor. Logic bits 0 and 1 can be determined by comparing the output current at a certain timestamp with a threshold. The simulation is carried out with three voltage levels: d) 2.8, e) 3.0, and f) 3.3 V. As the applied voltage increases, the mean switching time of memristor (λ) decreases.

device and can be scattered in a wide range. To systematically analyze the switching behavior of memristors, present works such as refs. [12,33,34] have conducted experiments by repeatedly performing the SET/RESET operation and statistically analyzing the resultant switching delays. They have revealed that the switching probability of a memristor model can be fitted into two distributions, the Poisson (Figure 2d–f) or the log-normal distribution. For a particular memristive device, the switching probability is yielded by the magnitude of the input voltage and the width of each pulse and correlated to the wait time until the switching event.

3.3. Gaussian RNG

GRNGs are the essential elements for performing Monte Carlo simulation, which constantly generates random numbers that are drawn from a standard normal distribution.^[35] To perform Monte Carlo simulation in solving SDEs, high-quality GRNs are required for generating the Wiener process. Existing hardware GRNG can be separated into four categories based on the algorithm they use, 1) cumulative distribution function (CDF) inversion, which infers GRNs from the standard Gaussian CDF;^[36,37] 2) transformation methods which convert uniform random numbers into GRNs with arithmetic manipulations;^[38,39] 3) acceptance–rejection methods which conditionally rejects some values after applying the transformation method;^[40] and 4) the recursion method which iteratively generates new GRNs through a pool of old GRNs.^[41] The first three methods involve the evaluation of mathematical expressions, such as sine, cosine,

and logarithms. These evaluations are expensive to be implemented on hardware, especially under power/area-limited scenarios. The recursion method, in contrast, requires memory for storing all the old GRNs but performs simply a linear combination of the old numbers which reduces the computational complexity. In the GRNG design of this work, we take advantage of the crossbar manner of memristors such that these transformations can be done within one clock cycle, hence saving time and power for generating GRNs for Monte Carlo simulation.

Another consideration of the GRNGs is the uniform random number generator (URNG) which is responsible for generating addresses of random numbers in memory. Commonly seen pseudo RNGs are derived from mathematical concepts, where the output is a deterministic function of the previous iteration.^[42] Though this deterministic property can be useful in some cases, the pseudo RNGs suffer from vulnerabilities such as repeated patterns.^[43] In this work, we use true RNGs built on memristors' stochastic switching behaviors to enhance the quality of randomness in the Gaussian paths generated. Existing work such as provided in refs. [44–46] each proposed a different memristor-based RNG structure, and the quality of the numbers generated had been proven with the National Institute of Standards and Technology test suite.

4. Proposed Architecture

Presented here is a general structure for iteratively solving SDEs within a hardware–software combined system using the

algorithm depicted in **Figure 3a**, which will be further explained later. An overall 16-bit fixed-point arithmetic (Q3.13), with 3 bits for the integer part and 13 bits for the fraction part, is used for representing the number manipulated within the computing system. The negative values in the crossbars are represented by the sum of a differential pair in the crossbar. For each number, one column V^+/I^+ represents the positive part and one column V^-/I^- represents the negative part and the number in voltage/current is represented by $V = V^+ - V^-$ or $I = I^+ - I^-$.

4.1. Technology Selection

The deterministic and stochastic solution paths to the SDE can each be evaluated by an iterative Euler solver and a GRNG, respectively. Previous work on Euler solvers or Monte Carlo simulators focused on either CMOS^[47,48] or fully analog implementations.^[49,50] These structures suffer from high energy consumption or large footprint area due to the serial multiply-and-accumulate (MAC) operations. Similarly, digital GRNGs have the same issue in time and energy cost when evaluating the expensive mathematical expressions with the Box–Muller^[38] or Ziggurat algorithms.^[40] Therefore, in this work, we propose a novel approach to building SDE solvers based on memristive devices, taking advantage of both the VMM operations in the crossbar for iterative MAC and intrinsic stochasticity for random number generation. Moreover, the devices can be manufactured on a nanometer scale with promising read speed which shortens the execution time and reduces the power and area consumption of the system.^[7,11]

4.2. General Structure

The solver structure is divided into two parts, in correspondence with the two terms in the SDE.

4.2.1. Stochastic Term

The stochastic term is generated by GRNs that follow a normal distribution with zero mean and standard deviation Δt (Figure 3b). The GRNs are generated by a memristor-based GRNG named WALLAX, iteratively. The structure uses the Wallace method^[51] to take advantage of the crossbar structure for linear transformations while harnessing the intrinsic stochastic nature of memristors for randomness. At the beginning of the process, a software algorithm such as the Box–Muller transform is used for generating an initial pool of N GRNs (Figure 3a, software part). These generated GRNs are normalized to achieve an average squared sum equal to one.^[51] The normalized values are stored in memory and randomly fetched per iteration. The fetching addresses are generated from a URNG^[44] which produces a bitstream with length $\log_2 N$ and is made up of three components, start, stride, and mask. The three components are transformed into database addresses by adding multiples of stride to start and XORed with a mask (i.e., $\text{add } r_i = (\text{start} + \text{stride} \times i) \oplus \text{mask}, i \in \{0, 1, 2, 3\}$). This approach ensures that the addresses are decorrelated and have minimal chance of overlapping.^[52] Accordingly, the hardware structure iteratively fetches the old-pool GRNs, organizes them into a vector, and multiplies the vector with a Hadamard matrix for linear transformation. New GRNs are collected as the output current flowing from each pair of differential columns. The new GRNs

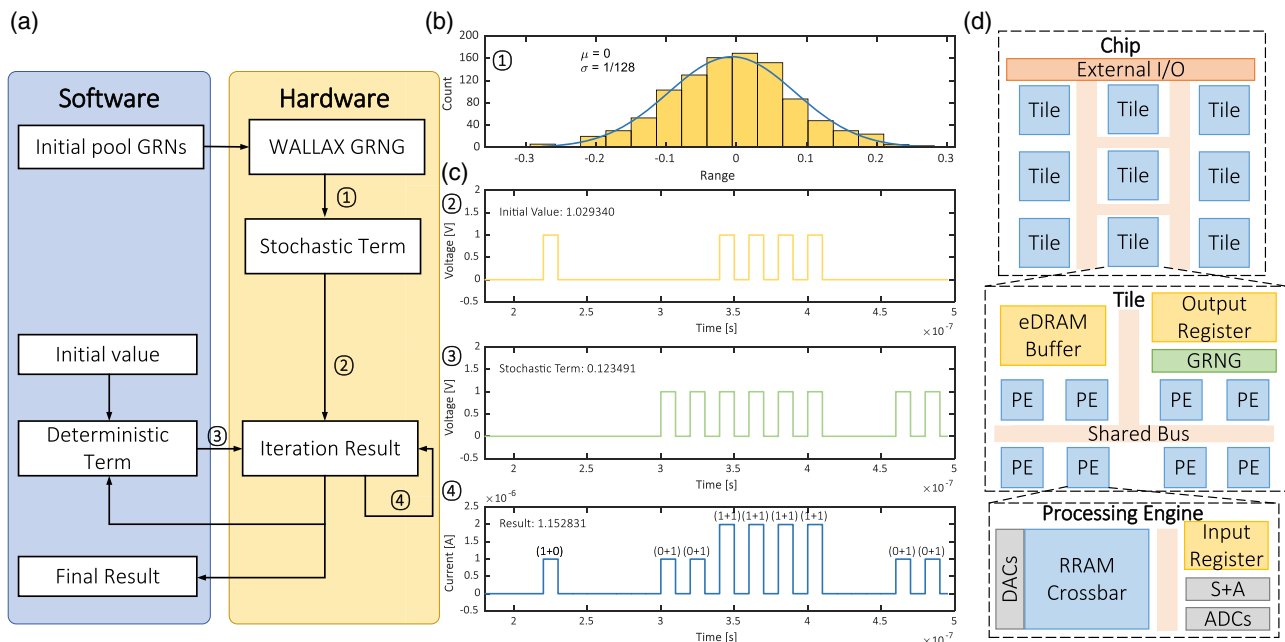


Figure 3. a) Workflow of the proposed stochastic differential equation (SDE) solver, with the function values evaluated in software and iterative paths calculated in hardware. b) The Gaussian random numbers (GRNs) generated, with zero mean and variance equals step size to be used as the stochastic term of the SDE solver. c) The initial value and stochastic term to be fed into the rows of crossbar and the resultant current sensed at the end of each column. d) The architecture hierarchy of the proposed SDE solver.

are written back into memory to allow for the recycling of numbers, eliminating the need for further generation of GRNs by software (Figure 3a). Before outputting to the peripherals, the new GRNs are corrected by a factor G where

$$G = \sqrt{\frac{S}{N}} = \sqrt{\frac{1}{2N}}(x + \sqrt{2N-1}) \quad (6)$$

to rewind the normalization process in the first step. The output Gaussian random vectors (ΔW) are multiplied with the coefficient functions and fed into the deterministic crossbar. The advantage of using such a system for random number generation is that it only requires software to generate one pool of initial GRNs and the subsequent values can be completely produced by hardware. Moreover, there can be multiple numbers generated per iteration, meaning that the random numbers can be each fed into a different deterministic term system to allow parallel computations.

4.2.2. Deterministic Term

The deterministic crossbar is responsible for summing the three components (X_n , $\mu\Delta t$, and $\sigma\Delta W$) in Equation (3). Each column $j \in \{1, 2, \dots, N\}$ of the crossbar represents the time-step $j\Delta t$ and has its value updated at the $(j-1)$ th iteration, N is the total number of steps, and the 0th iteration is a direct assignment of the initial value X_0 . The matrix to be mapped into the crossbar contains "1"s in the diagonal and superdiagonal elements, and "0"s in the rest. At each iteration i , the i th row and the $(i+1)$ th row are fed with the value of X_{i-1} and $\mu(t_n, X_n)\Delta t + \sigma(t_n, X_n)\Delta W$, respectively, as shown in Figure 3c. Only the sensing current at the $(i+1)$ th column is collected and directed to the $(i+1)$ th row for the next iteration. In a total of $(N-1)$ iterations, we are able to obtain the numerical solution path to the SDE. In most of the SDEs, the two coefficient functions, $\mu(t_n, X_n)$ and $\sigma(t_n, X_n)$, are nonlinear and can be merely evaluated on a normal computer. In case the functions are linear, they can also be evaluated on the crossbar, with the diagonal element still being "1," the super-diagonal elements mapped to the value of $\sigma(t_n, X, n)\Delta t$ and an additional set of sub-diagonal elements representing $\mu(t_n, X, n)\Delta t$.

4.2.3. Overall Structure

As shown in Figure 3d, the overall structure composes of one large memristor crossbar system for summing up the deterministic and stochastic term, an RNG containing two crossbars for simulating the Brownian motion and peripheral circuits including amplifiers, analog-to-digital converters (ADCs) and digital-to-analog converters (DACs). Memories are required for storing the random numbers and the solution paths before and after the computations.

4.3. Precision Extension

The precision of calculation is critical to the numerical solver of SDEs. Due to the iterative nature, inaccurate results in earlier steps can be accumulated and exaggerated in the later steps, leading to implausible results that are deviated from the actual

solution. Moreover, during each iteration, the numbers manipulated can pertain to multiple significant figures and result in a long bitstream. To ameliorate the potential loss in accuracy during calculation, a set of techniques has been adapted to the structure, targeting 16-bit fixed-point arithmetic. The precision extension method can be generalized as a two-step process, applied to the bitstreams in each calculation and the iteration steps for the overall solver. The first slicing determines the number of crossbars within each iteration group, and the second slicing determines the size of each crossbar and the number of iterations to be carried in each slice.

The number of quantized levels is correlated to the precision of the overall calculation since each device can only store a limited amount of bit width. For example, to perform a 16-bit calculation with a single memristor cell, it acquires a 16-bit DAC and 2^{16} resistance levels within the memristive device. As the number of resistance levels increases, the allowable range for each level decreases and the result obtained from the read operation can be deviated because of the nonidealities in the structure and inherent noises with the circuit. The need for peripheral devices handling over 16 bits also employs enormous overheads, resulting in large power consumption.

To ensure the exact computation, the numbers to be programmed onto an individual crossbar are limited to less than 4 bits. To generalize the process, we assume that each crossbar is designed to represent n bits. Thus, we need $l = \frac{16}{n}$ crossbars in parallel to perform the calculation, indexed from 0 to $l-1$. To perform the 16-bit calculation, a multiple crossbar system with a parallel connection is introduced to minimize the time taken for each step. The k th iteration's input X_i^k and deterministic constant terms C_i^k are sliced into bitstreams a_{ij} and c_{ij} starting from the least significant bit into the form

$$X_i^k = \{a_{i(l-1)}, \dots, a_{i2}, a_{i1}, a_{i0}\} \quad (7)$$

with first index $i \in \{0, 1, \dots, m-1\}$ representing the number from which the slice comes and the second index representing the crossbar index the slice feeds in and

$$C_i = \{c_{i(l-1)}, \dots, c_{i2}, c_{i1}, c_{i0}\} \quad (8)$$

with the same naming convention as the inputs. All the iteration inputs are then piled up into

$$\begin{aligned} \vec{X} &= \begin{bmatrix} a_{0(l-1)}, \dots, a_{01}, a_{00} \\ a_{1(l-1)}, \dots, a_{11}, a_{10} \\ \vdots \\ a_{(m-1)(l-1)}, \dots, a_{(m-1)1}, a_{(m-1)0} \end{bmatrix} \\ &= (\vec{X}_{l-1}, \dots, \vec{X}_1, \vec{X}_0), \end{aligned} \quad (9)$$

where $\vec{X}_{l-1}, \dots, \vec{X}_1, \vec{X}_0$ are n -bit vectors with m entries, and serve as inputs to each crossbar. Since the coefficient matrices for each iteration contain only 0 and 1 s (Figure 4a), there is no need to further slice them. The coefficient matrix and constant terms are then written to the conductance of memristors and the inputs are applied to the word lines of the crossbar with the corresponding index as seen in Figure 4b,c. The result of each slice is

$X'_{ij} = \{a'_{0j}, a'_{1j}, \dots, a'_{m-1j}\}$ where a'_{ij} is the corresponding crossbar output of input term a_{ij} , $j \in \{0, 1, \dots, l-1\}$. These results will be saved in registers and summed up in the end of each iteration by following the equation

$$X_i^{k+1} = a'_{i0} \times 2^{-\frac{8}{i}} + a'_{i1} \times 2^{-(\frac{8}{i}-1)} + \dots + a'_{i(l-1)} \times 2^{\frac{8}{i}} \quad (11)$$

where all the multiplications are realized by bit shifting.

When converting back to a floating point number after a fixed-point VMM operation is completed, the transform is reversed by rescaling the output vector and adding a constant vector with all identical elements containing the offset term. Figure 4d shows the solution paths generated from VMM with different slicing widths and compared to the analytical solution of the Black–Scholes equation.

5. Simulation

5.1. Methodology

The implementation of the Euler–Maruyama method on memristive crossbar is simulated on the MemTorch^[29] platform for its compatibility with different memristor models, that is, Stanford-PKU^[53] and voltage ThrEshold adaptive memristor (VTEAM)^[54] and the effect of different nonideal conditions, including the stuck rate of memristors and conductance variations. The line resistance of the crossbar, and ambient

temperature, in contrast, are examined on the passive crossbar simulation platform built from.^[55] All simulation parameters have been included in Table 1. The VTEAM model acts as a baseline for simulating the correctness of the proposed algorithm, while the Stanford-PKU model and the passive crossbar platform are used for nonideality tests. In all simulations, we take into account the device-to-device Ron/Roff variation by randomly sampling the resistance from a Gaussian distribution with a variance equal to 10% of its mean. Another form of device variation, switching time variation is not considered as all the programming of devices can be done prior to computations. The solution path generated from the hardware VMM technique is directly compared with the one that is generated by direct software VMM using the Euler–Maruyama method. In the nonideality test, to better visualize the effect, we calculated the absolute distance between the correct solution path and the generated path under nonideal conditions.

The simulation of a crossbar at a very large size could potentially last for a long time since the matrix is large in size and complex in the computations to be performed. To reduce the time elapsed, we developed another scheme that further slices the process to ensure that the size of the crossbar remains at a small scale. As shown in Figure 5e,f, for a solution path that contains 128 steps, we divided them into slices of three-step paths; hence, for each slice, we only need crossbars at the size of 3×3 . The last solution step in the last slice will be propagated to the initial value of the next slice such that the sliced paths can

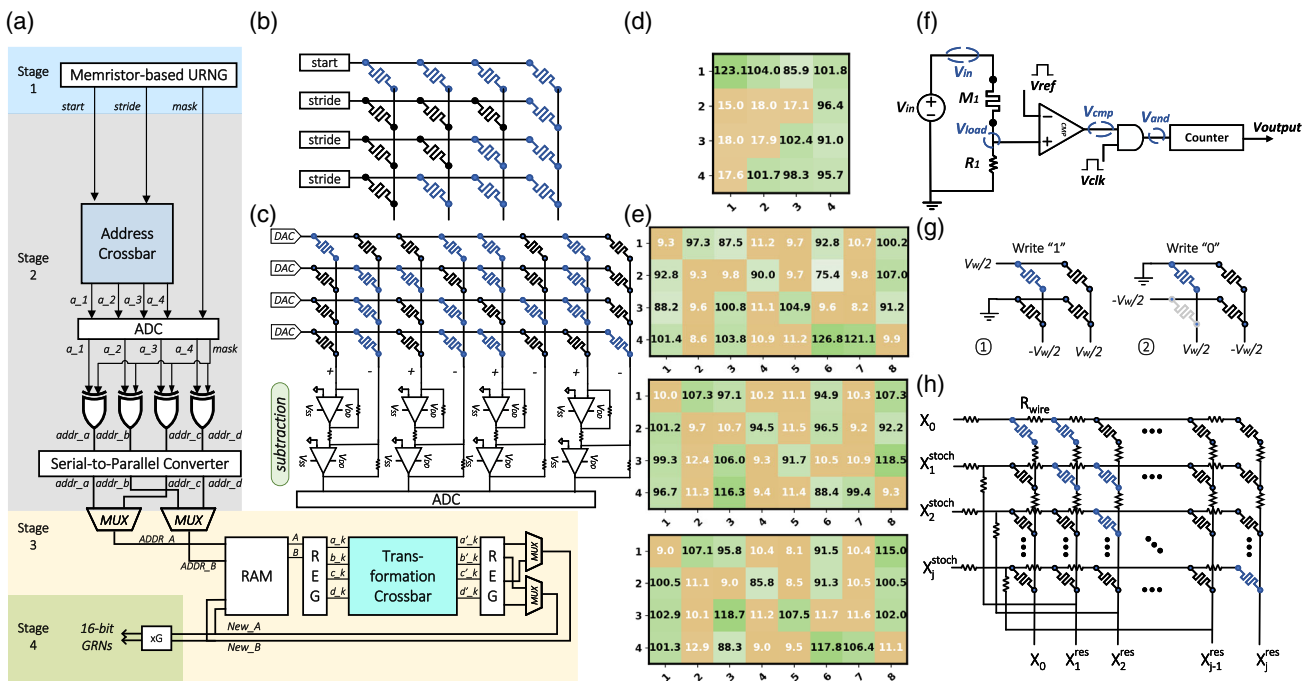


Figure 4. a) The overall Gaussian random number generator (GRNG) architecture that contains four stages: first two stages for random address generation and the third and fourth stage for GRN fetching and transformation. b) The address crossbar for accumulating the “strides” on the “start”. Black memristors are “OFF” and the blue memristors are “ON.” c) The transformation crossbar for linear combination of old GRNs. Each two columns of the crossbar are treated as differential pair and subtracted negative from positive to represent values. In actual simulation, the resistance of the d) address crossbar and e) the transformation matrix are marked in heatmaps. f) The memristive URNG structure that is proposed in ref. [44]. g) The write and verify process for simulating the mapping of matrices to the crossbar. h) The crossbar model for the deterministic term of the SDE solver with iterative and recursive calculation steps.

Table 1. Simulation parameters for SDE solver.

MemTorch				Passive			
Crossbar properties							
Ron [kΩ]	1	Ron [kΩ]		0.54		Ron [kΩ]	10
Roff [kΩ]	100	Roff [kΩ]		218.6		Roff [kΩ]	100
Ron/Roff variation range	10%	Ron/Roff variation range		10%		Ron/Roff variation range	10%
Wire resistance [Ω]	–	Wire resistance [Ω]		–		Wire resistance [Ω]	20
Stuck probability	1%	Stuck probability		1%		Stuck probability	1%
Memristor model parameters							
VTEAM				Stanford-PKU			
D	3.00 E-09	gap_init	2.00 E-10	t_ox	1.20 E-08	Vdd [V]	0.2
k_on	–10	g_0	2.50 E-10	F_min	1.40 E + 09	Resolution	2 bits
k_off	0.0005	V_0	0.25	vel_0	10	Bias_scheme	1/3
Alpha_on	3	I_0	0.001	E_a	0.6	Conductance model	linear
Alpha_off	1	Read_voltage	0.1	a_0	2.50 E-10		
v_on [V]	–0.2	T_init	298	Delta_g_init	0.02		
v_off [V]	0.02	R_th [Ω]	2100	Model type	Standard		
x_on	0	Gamma_init	16	T_crit [K]	450		
x_off	3.00 E-09	Beta	0.8	T_smth	500		

be concatenated into the solution path. As shown in Figure 5g, the runtime of simulation is reduced significantly by the reduction in slice size. The minimum mean absolute error of the 128-step path occurs when slice size = 8. This then becomes the size we kept for future simulations.

5.2. Validation of Correctness

To verify the correctness of matrix and operations mapped to the crossbar, we simulate the solver structure with an SDE that has analytical solution. The Black–Scholes asset pricing model^[23] is used for monitoring the stock exchange process and has the following form

$$dX(t) = \alpha X(t)dt + \beta X(t)dB(t), X(0) = X_0 \quad (12)$$

where $\alpha, \beta \in \mathbb{R}$. The model has an exact analytical solution

$$X(t) = X(0)e^{(\alpha - \frac{\beta^2}{2})t + \beta B(t)} \quad (13)$$

where $B(t)$ is the value of the Brownian motion path generated at time t . Euler–Maruyama Monte Carlo simulation is used in all cases to obtain the numerical solutions to the SDE with the initial value $X_0 = 1$ in 128 steps. The result obtained for each solution path is then compared with the analytical solution under the same stochastic process.

As shown in Figure 6a, the resultant numerical solution obtained from the proposed structure coincides with the solution path generated by software VMM using the same stochastic path and initial condition with the Stanford-PKU model.^[53] The averaged absolute difference between hardware and software VMM solution paths is 0.057. The performance of the system is also compared across memristor models with the same settings to test the compatibility of the structures. In Figure 6b, paths

generated by the two models have been recorded and compared visually. There is no significant observable difference between them. The mean absolute difference between the paths from the two models is 0.0508.

5.3. Solution Convergence

The final numerical solution to an SDE can be obtained by averaging the solution paths with different stochastic processes considered following the Monte Carlo simulation. In this section, we tested the Ornstein–Uhlenbeck process for the convergence of solution paths. The Ornstein–Uhlenbeck process^[56] is considered as a univariate continuous Markov process with its solution evolving with time.^[57] The process is defined over the state space $X \in \mathbb{R}$ with the SDE

$$dX(t) = -\frac{x - \mu}{\tau}dt + \sigma\sqrt{\frac{2}{\tau}}dB(t) \quad (14)$$

where $\mu = 10$ is the mean, $\sigma = 1$ is the standard deviation, and $\tau = 0.05$ is the time constant. The simulation results are shown in Figure 6c with the initial value of $X_0 = 0$. The gray-dashed lines depict the minimum/maximum values in all solution paths at each time step. We averaged all simulated solutions every 50 trials (Figure 6c) and observed that the resultant curve gradually converges in the center of the margin despite the perturbations introduced by the stochastic process.

5.4. Ambient Temperature Effect

The current–voltage (I – V) characteristic of the memristor is dependent on the temperature it operates in ref. [58]. In this section, the performance of the memristive solver is rendered

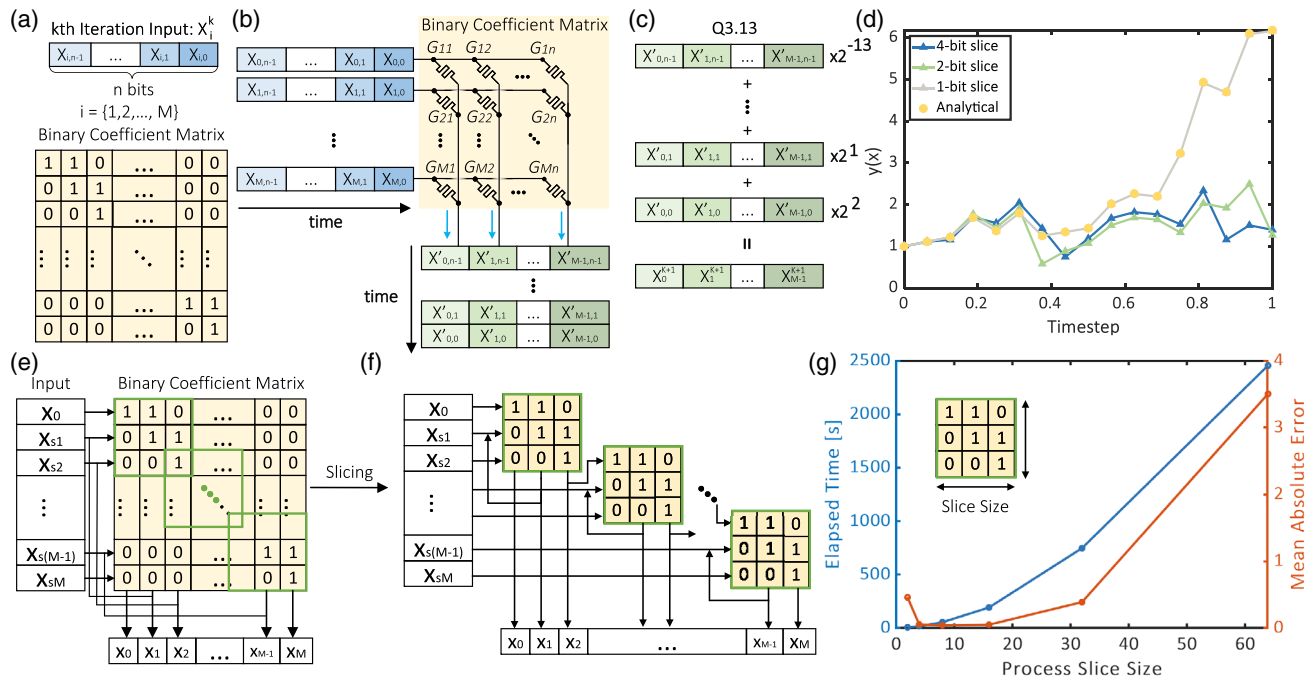


Figure 5. Precision extension methods: a) the input and matrix to be mapped to crossbar at each iteration. b) The slicing and collection of the input and output on the crossbar. c) The reconstruction of the output according to the bit-slicing scheme. d) The solution paths generated from VMM with different slicing widths and compared to the analytical solution of the Black–Scholes equation. The analytical path overlaps with the 1-bit sliced path. e) The iterative solver fluctuations without process slicing applied. f) The current flow of input and output of each crossbar with process slicing of size 3 applied. g) The fluctuation of mean absolute error of solution path and the elapsed time of simulation with different slice sizes on a 128-step solution path. The elapsed time decreases with the process slice size and the minimum error appears at slice size = 8.

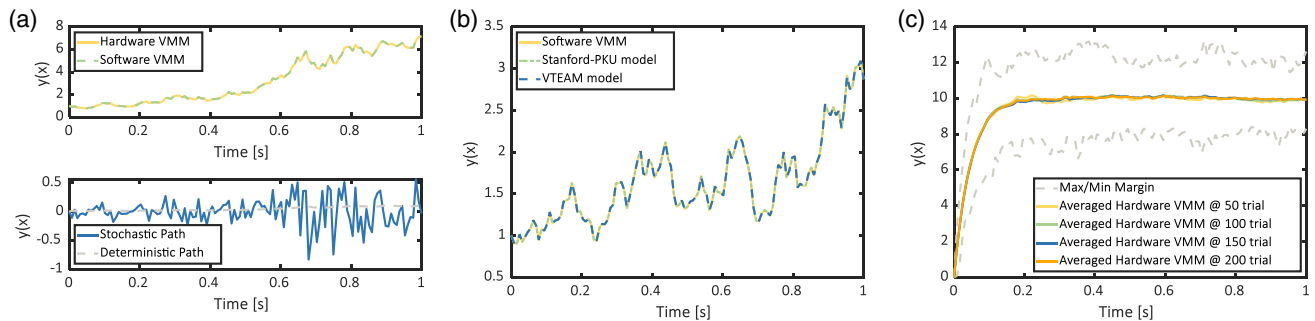


Figure 6. a) Result of the path correctness test: the upper figure contains the solution path generated by hardware and software VMM based on the Euler–Maruyama method and the lower figure plots the corresponding stochastic and deterministic path, respectively. b) The solution path generated by both VTEAM and Stanford-PKU model comparing to the software VMM path. c) The averaged solution path every 50 trials. The gray-dashed line indicated the minimum/maximum solution value at each time step.

at various temperature settings by calculating the mean-squared errors between the hardware solver path and the analytical software path per time step. As shown in Figure 7a,b, the root-mean-squared errors (RMSE) are recorded at every 10 Kelvin steps. The best proximity occurs when the ambient temperature is close to the room temperature at around 25 °C (298 K). The error increases consequently as the temperature further deviates from the room temperature with escalating magnitude. As the path length grows, the influence brought by temperature variation becomes more prominent. The increment between the minimum and maximum errors rises from 36.6% to 117.5%.

5.5. Device Variation Effect

The accuracy of inference on the crossbar can be influenced by nonidealities in the crossbar system, such as wire resistance, stuck probability of memristors, and the sneak-path current.^[29] The value collected from the crossbar inevitably deviates from the supposed correct value. In this section, we examine the extent of the error to which different crossbar nonidealities bring by showing the variation in RMSE for each path or time step as the nonideal situation varies. The simulations are run through a passive crossbar platform built upon.^[55] The platform assumes accurate programming of the memristors, such that we focus

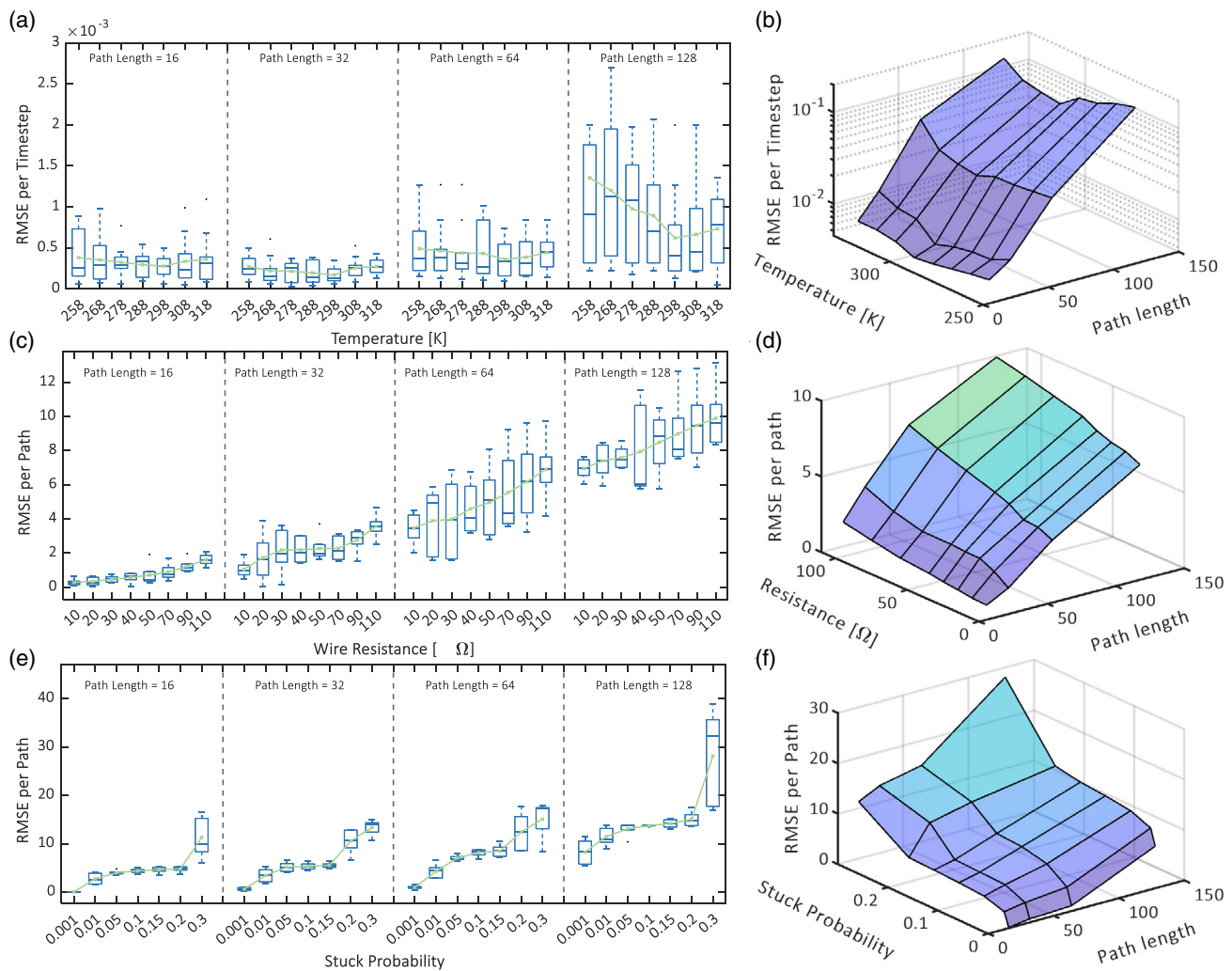


Figure 7. Nonideality test results: a,b) the variation of root-mean-squared errors (RMSE) with the ambient temperature in Kelvin and the solution path length. c,d) The variation of absolute error between the analytical and the hardware VMM path with the wire resistance in Ohm and the solution path length. e,f) The variation of RMSE with the change in memristor stuck probability and the solution path length.

on investigating how the nonideal conditions impact the computation accuracy. Thus, it is considered together with the wire resistance and stuck probability of the crossbar.

The wire resistance here is defined as the resistance between individual elements of the crossbar. Therefore, the total resistance of wires in a crossbar is proportional to the number of elements in it. Wire resistance degrades the signal passing through it and can potentially change the results. For testing the influence of various wire resistance ($[10, 110]\Omega$) on the accuracy of calculation, we calculated the absolute error ($|\text{crossbar}[i,j] - \text{software}[i,j]|$) between the crossbar solution and the software solution with an increasing number of steps in the solution path ($[16, 128]$). As shown in Figure 7c, the error increases proportionally to the wire resistance and path length from 0.2560 at ($10\Omega, 16$ steps) to 9.90 at ($110\Omega, 128$ steps). The magnitude of increment decreases as the wire resistance approaches the highest values for the same path length and as the path length increases with the same resistance settings as shown in Figure 7d with continuous scales.

Since we are only mapping binary numbers to the crossbar, the stuck state of memristors will lead to wrong calculation if the state is incorrect. The error further propagates through the path and results in exponentially growing errors. As shown in Figure 7e, the RMSE increases with the stuck probability and the path length, from 0.023 at (0.1%, 16 steps) to 28.11 at (30%, 128 steps). The impact of rising stuck probability is more significant than the wire resistance. Since the probability chosen is not equally spaced, we plot all the points together in Figure 7f and the trend shows that as the stuck probability and path length increase, the magnitude of increment in error also grows.

6. Results and Discussion

The reliability of the SDE solver system is tested against several applications. The quality of the numerical solution is evaluated based on the differences between the generated result and the

solution produced by software-based SDE solvers and analytic solutions when they exist. In the previous section, we demonstrated the reliability of the solver for solving some simple SDE problems. Here, we will demonstrate that the system can be used to solve a variety of interesting and useful problems, where the power and speed of our system will be of significant use.

6.1. First-Passage Time and Moments

To show the usefulness of our simulator for the common applications of stock prices modeled by geometric Brownian motion, we simulate 500 Black–Scholes trajectories with a VTEAM memristor model with a process slice size of 8, as shown in **Figure 8a,b**. We then calculate various statistics using these simulations.

The first-passage time or first hitting time of an SDE is roughly the amount of time it takes for a stochastic process to reach a threshold which may be time dependent. This concept is useful in the modeling of physical and economic systems, such as economic bubbles and credit defaults.^[59] Here, the stochastic process is the value of the stock price, and the first-passage time is some threshold when we may decide to sell the stock (or in the case of barrier options, the value of the stock depends on the time it takes to reach the barrier, if at all). We want to calculate the expected time until we can recoup our investment for this kind of contract. For a general time-dependent threshold and SDE, there may not exist a closed-form solution for the first-passage time, which would generally be found by solving the corresponding partial differential equation with appropriate boundary conditions. Here, we derive the first-passage time computationally using our hardware solution utilizing a VTEAM memristor

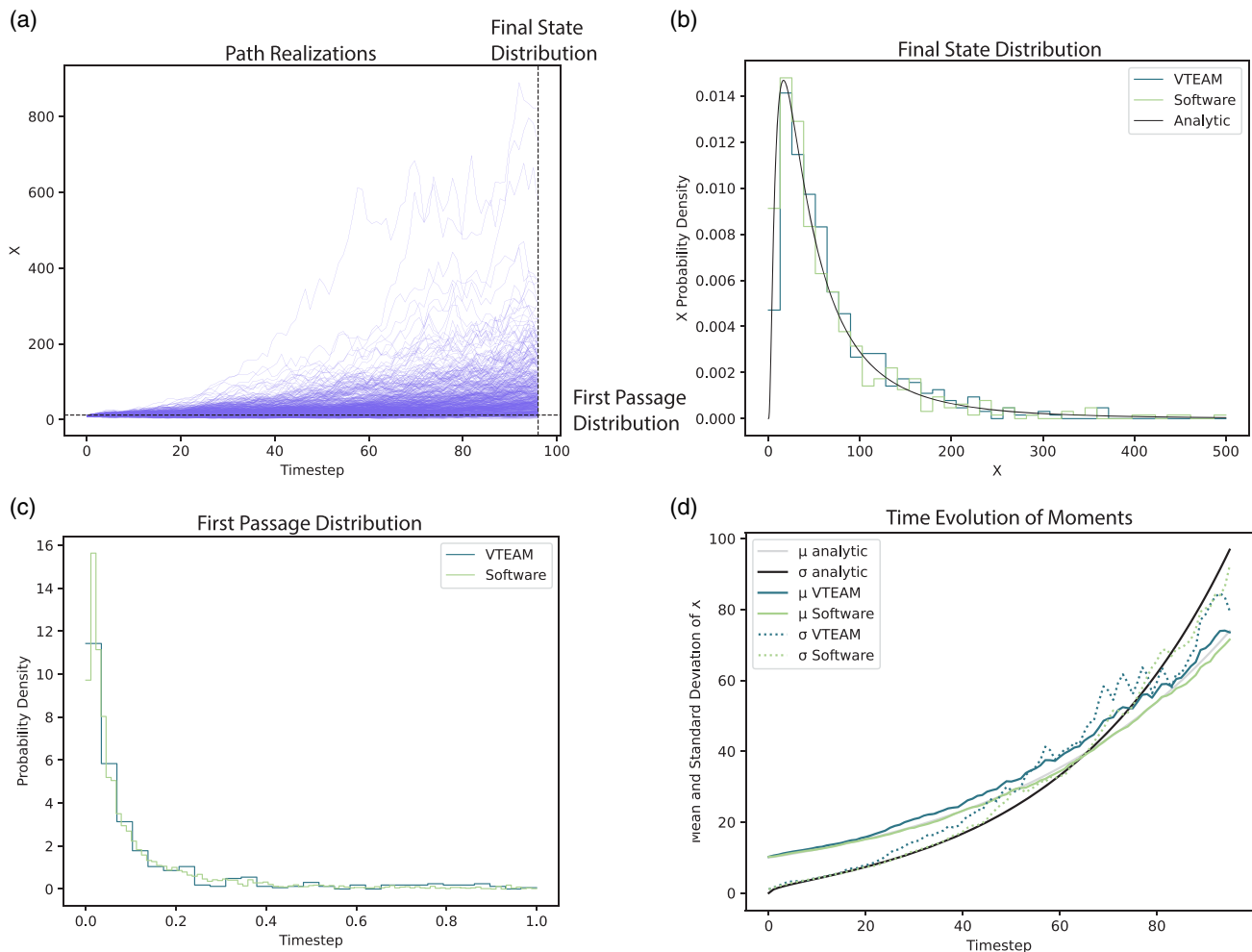


Figure 8. a) A total of 500 simulated Black–Scholes trajectories between times 0 and 1 with 96 timesteps using the VTEAM memristor model. The vertical line shows the slice of the joint probability density in time and in space that corresponds to the final state. The horizontal line shows the level with which the first-passage time is calculated. b) The final-state distribution at time 1. The analytic solution marginalized over all values of Brownian motion is plotted along with the software solution given the same realizations of noise using our WALLAX GRNG. The distributions are all in good agreement. c) The first-passage probability density with respect to the level $X = 12$, given that the process did reach the level. The VTEAM model simulation path is plotted along with a separate simulation on the software of 5000 paths with 20 000 timesteps. d) The mean and standard distribution of the VTEAM simulation, software, and analytic distributions.

model, as shown in Figure 8c, and compare it with a software simulation with a much finer time step (20 000) and more paths (5000). The distributions are in good agreement, indicating the utility of our hardware for performing first-passage computations.

We also compute the time evolution of the moments of the distribution by computing them for each timestep, as shown in Figure 8d. These computations are useful for example to value options and quantify risk. We also use them later to infer parameters of the stochastic process via stochastic gradient descent.

6.2. Stochastic Van der Pol Oscillator

To demonstrate the possible application of our proposed hardware solution in higher-dimensional nonlinear systems, we demonstrate a stochastically perturbed Van der Pol (Equation (15)) that uses a 1D noise process^[60]

$$d \begin{bmatrix} X \\ Y \end{bmatrix} = \begin{bmatrix} Y \\ \mu(1 - X^2)Y - X \end{bmatrix} + \lambda \begin{bmatrix} 0 \\ Y \end{bmatrix} dW \quad (15)$$

The simulations use the Stanford-PKU model with a process slice size of 8, and take 256 timesteps between times 0 and 30, with parameters $\mu = 4$ and $\lambda = 0.3$. We see that the system is capable of converging to the limit-cycle characteristic of the non-stochastic ODE, but the noise causes the spikes to happen at different times with a similar height (Figure 9a,b), and in fact raising the noise amplitude reduces the period of the spikes.

This model is useful for nonlinear circuits such as those containing memristors,^[61] and as a model for a neuron.^[62]

6.3. Fitting Parameters and Learning Dynamics of SDEs

As already mentioned, SDEs can be used to model the dynamics of physical systems. If the functional form of the physics is known, then the SDE can be discretized and techniques such as maximum likelihood can be used to infer the parameters of the model.^[63] However, if the functional form is not known, such as in the case of a highly nonlinear system, neural networks can be used to model f and g functions^[64] as shown in Figure 9c. This method is especially suitable for memristor crossbar hardware as it uses many linear computations and simple nonlinear computations to approximate general nonlinear functions.

As a simple first test of the utility of this method, we use our hardware solution to run forward passes of the Black–Scholes equation to fit the parameters of the SDE. The data was generated using 10 000 software solutions of the Black–Scholes equation starting from the same initial condition with 1000 time steps, which were interpolated into 30 length time series to ensure minimal error due to the inaccuracy of the Euler–Maruyama solver that generated them.

The hardware solver was then used to generate 30 solution paths with 30 time steps. The errors and weight updates were calculated in software and used to update the mean and variance. The hyper-parameters were chosen by hand to ensure reasonable convergence. The search is shown in Figure 9d, depicting the path in parameter space toward the actual μ and σ which

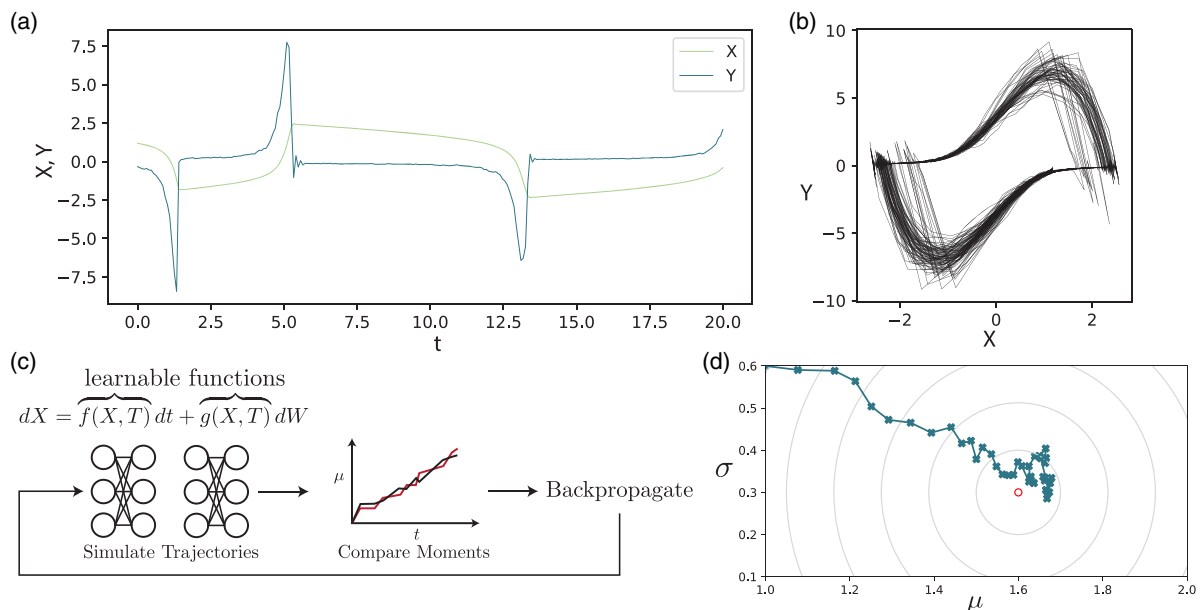


Figure 9. a) One sample trajectory of the stochastic Van der Pol oscillator. b) Phase portrait of the stochastic process showing all 48 paths. The noise causes slightly different paths through phase space. c) Scheme under the Sparse, Physics-based, and partially Interpretable Neural Networks framework^[64] for training learnable functions to approximate stochastic dynamics. In our problem, we fix the functional form and learn the parameters; however, the functions could be learned as well by parameterizing them as neural networks. d) The evolution of the Black–Scholes parameter estimates. The initial guess is (1.0, 0.6), starting in the top left corner. The solution converges to the true parameters that generated the data, shown by a red dot. The solution does not quite converge to the true answer due to the noise in the simulation, but increasing the number of paths that the hardware solver generates and the amount of data allows the search to converge closer to the true answer.

generated the data. The search does not quite converge to the true solution due to the coarseness of the hardware trajectories and their small number. Increasing them slows down the speed of the simulation but ensures better convergence. Figure 9c shows the trajectories of the mean and variance using the current estimate of the SDE parameters.

7. Analysis and Comparison

7.1. Resource Consumption Estimation

The resource requirement of the proposed SDE solver, including power, area, and latency consumption, is estimated based on the following assumptions: a constant reading voltage of $0.3 \text{ V}^{[65]}$ is used, and the digital peripherals are scaled to 32 nm technology with factors introduced in ref. [66]. Each memristor device has a

read latency of $6 \text{ ns}^{[20]}$ and a fixed area.^[67] Other circuits and crossbar parameters are consistent with the simulation settings as indicated in Table 1 and the path lengths are fixed to 128 time steps. Since the matrices mapped to the crossbars are fixed, all programming can be done ahead of calculations performed on the crossbar. Therefore, the cost of device programming is not considered in resource consumption.

The estimation of the resource consumption of the proposed structure has been broken down in Table 2 according to the two-term SDE solver structure proposed. The stochastic term is simulated by a GRNG which contains a total of 51 memristors (one 4×4 crossbar, one 4×8 crossbar, and 3 for RNG), 9 ADCs^[67] each with 8-bit resolution and 8 DACs^[67] with 1-bit resolution as a result of the accuracy enhancement method applied. The deterministic term is considered without any accuracy enhancement method applied. Both ADC^[68] and

Table 2. Resource consumption estimation of SDE solver without slicing.

Stochastic term—GRNG					
Component	Params	Specification	Power [mW]	Area [mm ²]	Latency [μs] ^{a)}
ADC	Resolution	8 bits	18	0.0108	0.00166
	Number	9			
	Frequency	1.2 GHz			
DAC	Resolution	1 bit	0.0313	1.33 E-06	1.60 E-03
	Number	8			
Transistor	Number	3	0.03	3.10 E-05	23.333
Comparator	Number	3	0.542	5.83 E-05	
Counter	Number	3	0.0321	1.68 E-03	
Exclusive-OR gates	Number	4	0.0207	1.78 E-05	8.20 E-05
Serial-to-parallel converter	Number	4	0.126	2.02 E-02	0.01
Subtractor	Number	4	5.6	2.12 E-03	5.00 E-06
Multiplexer	Number	8	2.40 E-01	5.21 E-05	2.25 E-05
Dual-port random-access memory	Size	2KB	4.59	2.48 E-03	8.21 E-05
Input register (IR) + output register (OR)	Size	256B	0.23	7.70 E-04	8.21 E-05
Memristors	Size	51	0.08958	1.07 E-04	2.00 E-02
	Bit per cell	2			
Total			29.53	0.0383	23.41
Deterministic term—processing engine (no slicing)					
Component	Params	Specification	Power [mW]	Area [mm ²]	Latency [μs] ^{a)}
ADC	Resolution	16 bits	81.024	66.56	0.1
	Number	128			
	Frequency	10 MHz			
DAC	Resolution	16 bits	2150.4	67.84	1.00
	Number	128			
S + A	—	—	—	—	—
IR + OR	Size	1 KB	0.674	8.10 E-01	8.21 E-05
Memristors	Size	128 × 128	69.648	1.15 E-03	6.00 E-03
	Bit per cell	2			
Total			2301.75	135.21	141.57

^{a)}Latency of individual component.

DAC^[69] in this case are considered with 16-bit resolution, no digital shift and add (S + A) blocks are required. For a total of 128 iterations, the proposed deterministic term solver consumes 2301.75 mW of power and 135.21 mm² area, while taking 141.57 μ s for the whole execution. The stochastic term is aimed at generating 128 random numbers either before the execution of the deterministic term or in parallel with it. In the former case, the stochastic term generates 4 GRNs per 23.41 μ s which results in a total of 749.12 μ s for generating all the GRNs as needed. All the random numbers and the equation parameters are stored in an embedded dynamic random-access memory buffer^[67] with a size of 64KB in the tile.

Table 3 records the estimated resource consumption of the proposed SDE solver with no, as well as two different accuracy enhancement techniques applied. The bit slicing method reduces the resolution requirement of both ADC and DAC, which now requires only 15.45^[70] and 8 mW^[67] of power, respectively. The matrix slicing also effectively reduces the total area

consumed by memristors by 94% as only the diagonal and super-diagonal elements remained in the latter approach.

7.2. Comparison Against Iterative Differential Equation Solvers

Many of the differential equation solutions are approximated by iterative solvers such as Euler and Runge–Kutta. These methods start from time step 0 and progressively evaluate and update the value of the following time step. In this section, we compare the performance of the proposed hardware with similar iterative solvers that are implemented on either CMOS,^[47,48,71,72] or fully analog computing.^[73,74] These works have shed light on enhancing the accuracy of numerical solvers to differential equations while achieving a minimal execution time. In **Table 4**, we compare the performance of our proposed memristive crossbar SDE solver with the mentioned literature using similar metrics.

Table 3. Resource consumption of the deterministic term with different enhancement techniques applied.

Deterministic term (no slicing)						Deterministic term (bit slicing only)				Deterministic Term (Bit and Matrix Slicing)			
Component	Params	Specification	Power [mW]	Area [mm ²]	Latency [μ s]	Specification	Power [mW]	Area [mm ²]	Latency [μ s]	Specification	Power [mW]	Area [mm ²]	Latency [μ s]
ADC	Resolution	16 bits	81.024	66.56	0.1	8 bits	15.450	1.5744	0.02	8 bits	15.450	1.5744	0.02
	Number	128				128				128			
	Frequency	10 MHz				50 MHz				50 MHz			
DAC	Resolution	16 bit	2150.4	67.84	1.00	1 bit	8.00	2.13 E-05	1.60 E-03	1 bit	8.00	2.13 E-05	1.60 E-03
	Number	128				128				128			
S + A	–	–	–	–	–	128	34.12	0.02	8.33 E-04	128	34.12	0.02	8.33 E-04
IR + OR	Size	1KB	0.674	8.10 E-01	8.21 E-05	1KB	0.674	8.10 E-01	8.21 E-05	1KB	0.674	8.10 E-01	8.21 E-05
Memristors	Size	128 \times 128	69.648	1.15 E-03	6.00 E-03	128 \times 128	69.648	1.15 E-03	6.00 E-03	8 \times 8	1.088	7.17 E-05	6.00 E-03
	Bit per cell	2				2				2			
Total			2301.75	135.2111	141.57	Total	127.89	2.4079	56.5383	Total	59.33	2.4086	56.5383

Table 4. Comparison between SDE solver and other iterative solvers.

Design	[47]	[48]	[71]	[72]	[73,74]	This work
Equation	ODE	ODE	ODE	SDE	ODE/PDE/SDE	SDE
Technology	CMOS (28 nm)	CMOS (Xilinx Virtex-6)	ASIC	CMOS (Altera Stratix)	Analog + CMOS (65 nm)	CMOS (32 nm) + resistive random-access memory
Method	Stochastic computing	Forward/modified Euler Runge–Kutta schemes	Runge–Kutta	Weak approximation Monte Carlo	Euler's method	Euler–Maruyama Monte Carlo
Bit length	8	32/64	–	32	–	16
Arithmetic	Floating point	Floating point	Fixed point	Floating point	Fixed/floating point	Fixed point
Hardware performance						
Path/block size	256	60	10 000	–	256	256
Best Accuracy (RMSE)	3.88 E-03	1.85 E-05	–	–	1/256	0.0072
Best execution time [ms]	1.05 E-04	0.52 (CPU double FP)	11	8 E-06 (per number) ^{a)}	0.2 ^{a)}	5.66 E-02 ^{b)}

^{a)}Estimated from given result. ^{b)}Considered only the iterative deterministic solver.

The RMSE of the proposed SDE solver, though very small, is still higher than the digital counterparts. This is due to the non-idealities in the crossbar during the simulation and the limited precision brought by fixed-point arithmetic. The truncated precision brings an approximately 0.05% error to each number without consideration of crossbar nonidealities. The iterative deterministic part of our SDE solver also has an execution time lying between the benchmark works. The most delay resulted from the DAC which is associated with the bit length and the path length, and in trade-off with the total power consumption.^[20] In this work, we selected existing DAC technology that has equally well performance in power consumption and sampling frequency. Since the power consumptions of the benchmarked iterative solutions are mostly unstated, it is hard to evaluate the performance of the proposed SDE solver in terms of both power and time consumption.

The solvers we compared report on the accuracy of their designs using different metrics. The accuracy is also influenced by the numerical method implemented, the arithmetic, and the hardware platforms used. Since we propose a novel architecture for iteratively and numerically solving the SDEs, the comparison table is aimed at providing information on how our design trades off between execution time and accuracy compared with existing work.

8. Conclusion

In this article, we proposed a memristor-based SDE solver that takes advantage of the inherent stochasticity and crossbar operation of memristors. It iteratively computes the numerical solution to the SDEs based on the Euler–Maruyama method and generates the final result with Monte Carlo simulation. The proposed system was tested on the correctness of computation and the tolerance of device nonidealities. The presented results showed that the proposed solver generates paths in high proximity to the software-generated analytical path despite the inaccuracy caused by extreme nonideal situations. The reliability of the proposed system was also tested with various application scenarios, and its capability of providing a great solution to practical problems is demonstrated despite the limited simulation speed and inaccuracy from fixed-point representation. For future work, the accuracy of the SDE solver can be further enhanced by floating point arithmetic and the solver itself can be expanded to adapt to multidimensional SDE systems where no analytical solutions exist and the acceleration effect of crossbars could be more significant.

Acknowledgements

A.A. acknowledges financial support by Natural Sciences and Engineering Research Council of Canada (DGECR-2022-00101). [Correction added on August 21, 2023, after first online publication: In the abstract, the sentence “Herein, the tolerance of the system is also accessed to crossbar nonidealities by comparing the numerical and analytical paths’ variation in error” was corrected to “The tolerance of the system to crossbar nonidealities is also accessed by comparing the numerical and analytical paths’ variation in error.”]

Conflict of Interest

The authors declare no conflict of interest.

Data Availability Statement

The data that support the findings of this study are available from the corresponding author upon reasonable request.

Keywords

Black–Scholes equation, Brownian path, memristor crossbar, stochastic differential equation, vector–matrix multiplication

Received: January 9, 2023

Revised: March 8, 2023

Published online: April 18, 2023

- [1] M. Bayram, T. Partal, G. Orucova Buyukoz, *Adv. Differ. Equations* **2018**, 2018, 1.
- [2] Y. Song, J. Sohl-Dickstein, D. P. Kingma, A. Kumar, S. Ermon, B. Poole, *ArXiv* **2021**, abs/2011.13456.
- [3] T. Szabados, *Stud. Sci. Math. Hung.*, **2010**, 53, 93.
- [4] H. Gilsing, T. Shardlow, *J. Comput. Appl. Math.* **2007**, 205, 1002.
- [5] G. Ansmann, *Chaos: Interdiscip. J. Nonlinear Sci.* **2018**, 28, 043116.
- [6] R. Nair, *Proc. IEEE* **2015**, 103, 1331.
- [7] M. Rahimi Azghadi, Y.-C. Chen, J. Eshraghian, J. Chen, C.-Y. Lin, A. Amirsoleimani, A. Mehonic, A. Kenyon, B. Fowler, J. Lee, Y.-F. Chang, *Adv. Intell. Syst.* **2020**, 2, 1900189.
- [8] D. Ielmini, H.-S. P. Wong, *Nat. Electron.* **2018**, 1, 333.
- [9] A. Amirsoleimani, F. Alibart, V. Yon, J. Xu, M. Pazhouhandeh, S. Ecoffey, Y. Beilliard, R. Genov, D. Drouin, *Adv. Intell. Syst.* **2020**, 2, 2000115.
- [10] M. R. Azghadi, C. Lammie, J. K. Eshraghian, M. Payvand, E. Donati, B. Linares-Barranco, G. Indiveri, *IEEE Trans. Biomed. Circuits Syst.* **2020**, 14, 1138.
- [11] C. Lammie, J. K. Eshraghian, W. D. Lu, M. R. Azghadi, *IEEE Trans. Circuits Syst. II Express Briefs* **2021**, 68, 1650.
- [12] S. Gaba, P. Knag, Z. Zhang, W. Lu, in *2014 IEEE Int. Symp. Circuits and Systems (ISCAS)*, IEEE, Piscataway, NJ **2014**, pp. 2592–2595.
- [13] L. Primeau, A. Amirsoleimani, R. Genov, in *2022 IEEE Int. Symp. on Circuits and Systems (ISCAS)*, Austin, TX, USA **2022**, pp. 1867–1871, <https://doi.org/10.1109/ISCAS48785.2022.9937861>.
- [14] C. Lammie, M. R. Azghadi, in *2019 IEEE Int. Symp. Circuits and Systems (ISCAS)*, IEEE, Piscataway, NJ **2019**, pp. 1–5.
- [15] F. Zahari, E. Pérez, M. K. Mahadevaiah, H. Kohlstedt, C. Wenger, M. Ziegler, *Sci. Rep.* **2020**, 10, 1.
- [16] S. Lv, J. Liu, Z. Geng, *Adv. Intell. Syst.* **2021**, 3, 2000127.
- [17] X. Dong, A. Amirsoleimani, M. R. Azghadi, R. Genov, in *2022 IEEE Inter. Conf. Omni-layer Intelligent Systems (COINS)*, IEEE, Piscataway, NJ **2022**, pp. 1–5.
- [18] M. A. Zidan, H. A. H. Fahmy, M. M. Hussain, K. N. Salama, *Microelectron. J.* **2013**, 44, 176.
- [19] C. Yakopcic, T. M. Taha, in *2013 Int. Joint Conf. Neural Networks (IJCNN)*, Dallas, TX, USA **2013**, pp. 1–8.
- [20] M. A. Zidan, Y. Jeong, J. Lee, B. Chen, S. Huang, M. J. Kushner, W. D. Lu, *Nat. Electron.* **2018**, 1, 411.
- [21] N. G. Van Kampen, *Phys. Rep.* **1976**, 24, 171.
- [22] P. Kidger, J. Foster, X. Li, H. Oberhauser, T. Lyons, in *Int. Conf. Machine Learning, virtual* **2021**.
- [23] F. Black, M. Scholes, *World Scientific Reference on Contingent Claims Analysis in Corporate Finance: Volume 1: Foundations of CCA and Equity Valuation*, World Scientific, Singapore **2019**, pp. 3–21.
- [24] T. Sauer, *Handbook of Computational Finance*, Springer, Berlin/New York **2012**, pp. 529–550.

- [25] K. K. Andersen, H. Madsen, L. H. Hansen, *Energy Build.* **2000**, 31, 13.
- [26] A. Saarinen, M.-L. Linne, O. Yli-Harja, *PLOS Comput. Biol.* **2008**, 4, e1000004.
- [27] X. Liu, T. Xiao, S. Si, Q. Cao, S. Kumar, C.-J. Hsieh, *arXiv preprint arXiv:1906.02355*, **2019**.
- [28] X. Li, T.-K. L. Wong, R. T. Q. Chen, D. Duvenaud, in *Int. Conf. Artificial Intelligence and Statistics*, IEEE, Piscataway, NJ **2020**.
- [29] C. Lammie, W. Xiang, B. Linares-Barranco, M. R. Azghadi, *Neurocomputing* **2020**, 485, 124.
- [30] Z. Wang, C. Li, P. Lin, M. Rao, Y. Nie, W. Song, Q. Qiu, Y. Li, P. Yan, J. P. Strachan, N. Ge, *Nat. Mach. Intell.* **2019**, 1, 434.
- [31] C. Li, D. Belkin, Y. Li, P. Yan, M. Hu, N. Ge, H. Jiang, E. Montgomery, P. Lin, Z. Wang, W. Song, *Nat. Commun.* **2018**, 9, 2385.
- [32] D. B. Strukov, G. S. Snider, D. R. Stewart, R. S. Williams, *Nature* **2008**, 453, 80.
- [33] S. H. Jo, K.-H. Kim, W. Lu, *Nano Lett.* **2009**, 9, 496.
- [34] G. Medeiros-Ribeiro, F. Perner, R. Carter, H. Abdalla, M. D. Pickett, R. S. Williams, *Nanotechnology* **2011**, 22, 095702.
- [35] J. Malik, A. Hemani, *ACM Comput. Surv.* **2016**, 49, 1.
- [36] R. C. Cheung, D.-U. Lee, W. Luk, J. Villasenor, *IEEE Trans. Very Large Scale Integr. VLSI Syst.* **2007**, 15, 952.
- [37] J. Chen, J. Moon, K. Bazargan, *IEEE Trans. Magn.* **2004**, 40, 1744.
- [38] G. Box, M. Muller, *Ann. Math. Stat.* **1958**, 29, 610.
- [39] N. Kasiviswanathan, K. Srivatsan, in *2017 Int. Conf. Nextgen Electronic Technologies: Silicon to Software (ICNETS2)*, Chennai, India **2017**, pp. 327–331.
- [40] W. Tsang, G. Marsaglia, *J. Stat. Software* **2000**, 05, 1.
- [41] C. Wallace, *ACM Trans. Math. Software (TOMS)* **1996**, 22, 119.
- [42] A. Marghescu, P. Svasta, in *2015 IEEE 21st Int. Symp. Design and Technology in Electronic Packaging (SIITME)*, IEEE, Piscataway, NJ **2015**, pp. 319–322.
- [43] F. James, L. Moneta, *Comput. Software Big Sci.* **2020**, 4, 1.
- [44] H. Jiang, D. Belkin, S. Savel'ev, S. Lin, Z. Wang, Y. Li, S. Joshi, R. Midya, C. Li, M. Rao, M. Barnell, *Nat. Commun.* **2017**, 8, 882.
- [45] S. Balatti, S. Ambrogio, R. Carboni, V. Milo, Z. Wang, A. Calderoni, N. Ramaswamy, D. Ielmini, *IEEE Trans. Electron Devices* **2016**, 63, 2029.
- [46] K. S. Woo, J. Kim, J. Han, J. M. Choi, W. Kim, C. S. Hwang, *Adv. Intell. Syst.* **2021**, 3, 2100062.
- [47] S. Liu, J. Han, in *2017 54th ACM/EDAC/IEEE Design Automation Conf. (DAC)*, IEEE, Piscataway, NJ **2017**, pp. 1–6.
- [48] I. Stamoulis, M. Möller, R. Miedema, C. Strydis, C. Kachris, D. Soudris, in *Proc. 8th Int. Symp. Highly Efficient Accelerators and Reconfigurable Technologies*, ser. HEART2017, Association for Computing Machinery, New York, NY **2017**.
- [49] G. Cowan, R. Melville, Y. Tsividis, *IEEE J. Solid-State Circuits* **2006**, 41, 42.
- [50] Y. Huang, N. Guo, M. Seok, Y. Tsividis, S. Sethumadhavan, *IEEE Micro* **2017**, 37, 30.
- [51] R. Brent, *Comput. J.* **2010**, 51, 579.
- [52] D.-U. Lee, W. Luk, J. Villasenor, G. Zhang, P. Leong, *IEEE Trans. Very Large Scale Integr. VLSI Syst.* **2005**, 13, 911.
- [53] Z. Jiang, Y. Wu, S. Yu, L. Yang, K. Song, Z. Karim, H.-S. P. Wong, *IEEE Trans. Electron Devices* **2016**, 63, 1884.
- [54] S. Kvatinsky, M. Ramadan, E. G. Friedman, A. Kolodny, *IEEE Trans. Circuits Syst. II: Express Briefs* **2015**, 62, 786.
- [55] A. Chen, *IEEE Trans. Electron Devices* **2013**, 60, 1318.
- [56] G. E. Uhlenbeck, L. S. Ornstein, *Phys. Rev.* **1930**, 36, 823.
- [57] D. T. Gillespie, *Phys. Rev. E* **1996**, 54, 2084.
- [58] J. Singh, B. Raj, *Eng. Sci. Technol. Int. J.* **2018**, 21, 862.
- [59] C. Yi, *Quant. Finance* **2010**, 10, 957.
- [60] H. Leung, *Phys. A* **1995**, 221, 340, proceedings of the Second IUPAP Topical Conference and the Third Taipei International Symposium on Statistical Physics.
- [61] M. Ignatov, M. Hansen, M. Ziegler, H. Kohlstedt, *Appl. Phys. Lett.* **2016**, 108, 084105.
- [62] P. Chorbani, S. Ramakrishnan, A. Whitman, H. Ashrafioun, *Biomed. Signal Process. Control* **2015**, 15, 1.
- [63] A. W. Lo, Working Paper 59, National Bureau of Economic Research **1986**.
- [64] J. O'Leary, J. A. Paulson, A. Mesbah, *J. Comput. Phys.* **2022**, 468, 111466.
- [65] W. Shim, Y. Luo, J.-S. Seo, S. Yu, in *2020 IEEE Int. Reliability Physics Symp. (IRPS)*, IEEE, Piscataway, NJ **2020**, pp. 1–5.
- [66] S. Sarangi, B. Baas, in *2021 IEEE Int. Symp. Circuits and Systems (ISCAS)*, IEEE, Piscataway, NJ **2021**, pp. 1–5.
- [67] A. Shafiee, A. Nag, N. Muralimanohar, R. Balasubramanian, J. P. Strachan, M. Hu, R. S. Williams, V. Srikumar, *SIGARCH Comput. Archit. News* **2016**, 44, 14.
- [68] Y.-H. Chung, C.-H. Tien, Q.-F. Zeng, in *2019 IEEE Asia Pacific Conf. Circuits and Systems (APCCAS)*, IEEE, Piscataway, NJ **2019**, pp. 5–8.
- [69] Y. H. Chung, C. H. Tien, Q. F. Zeng, *IEEE Trans. Circuits Syst. I Regul. Pap.* **2022**, 69, 88.
- [70] Z. Chen, M. Miyahara, A. Matsuzawa, in *2015 Symp. VLSI Circuits (VLSI Circuits)*, Daegu, Korea **2015**, pp. C64–C65.
- [71] C. Huang, F. Vahid, T. Givargis, *IEEE Embedded Syst. Lett.* **2011**, 3, 113.
- [72] F. Martini, M. Piccardi, N. Liberati, E. Platen, in *2005 IEEE Int. Symp. Circuits and Systems*, Vol. 2, IEEE, Piscataway, NJ **2005**, pp. 1702–1705.
- [73] Y. Huang, N. Guo, M. Seok, Y. Tsividis, S. Sethumadhavan, in *2016 ACM/IEEE 43rd Annual Int. Symp. Computer Architecture (ISCA)*, IEEE, Piscataway, NJ **2016**, pp. 570–582.
- [74] Y. Huang, N. Guo, S. Sethumadhavan, M. Seok, Y. Tsividis, in *2018 IEEE 23rd Int. Conf. Digital Signal Processing (DSP)*, IEEE, Piscataway, NJ **2018**, pp. 1–5.