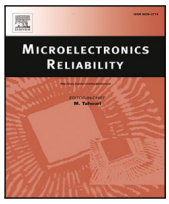
Microelectronics Reliability 168 (2025) 115630

Contents lists available at ScienceDirect

Microelectronics Reliability

journal homepage: www.elsevier.com/locate/microrel



Research paper

# RRAMulator: An efficient FPGA-based emulator for RRAM crossbar with device variability and energy consumption evaluation

Jianan Wen [a,*], Fabian Luis Vargas [a], Fukun Zhu [a], Daniel Reiser [b], Andrea Baroni [a], Markus Fritscher [a,c], Eduardo Perez [a,c], Marc Reichenbach [b], Christian Wenger [a,c], Milos Krstic [a,d]

[a] IHP - Leibniz-Institut für innovative Mikroelektronik, Frankfurt (Oder), Germany
[b] University of Rostock, Rostock, Germany
[c] Brandenburg University of Technology Cottbus-Senftenberg, Cottbus, Germany
[d] University of Potsdam, Potsdam, Germany

ARTICLE INFO

*Keywords:*
Resistive RAM
FPGA
Hardware Emulation
RRAM Crossbar
In-Memory Computing
Reliability

ABSTRACT

The in-memory computing (IMC) systems based on emerging technologies have gained significant attention due to their potential to enhance performance and energy efficiency by minimizing data movement between memory and processing unit, which is especially beneficial for data-intensive applications. Designing and evaluating systems utilizing emerging memory technologies, such as resistive RAM (RRAM), poses considerable challenges due to the limited support from electronics design automation (EDA) tools for rapid development and design space exploration. Additionally, incorporating technology-dependent variability into system-level simulations is critical to accurately assess the impact on system reliability and performance. To bridge this gap, we propose RRAMulator, a field-programmable gate array (FPGA) based hardware emulator for RRAM crossbar array. To avoid the complex device models capturing the nonlinear current–voltage (IV) relationships that degrade emulation speed and increase hardware utilization, we propose a device and variability modeling approach based on device measurements. We deploy look-up tables (LUTs) for device modeling and use the multivariate kernel density estimation (KDE) method to augment existing data, extending data variety and avoiding repetitive data usage. The proposed emulator achieves cycle-accurate, real-time emulations and provides information such as latency and energy consumption for matrix mapping and vector–matrix multiplications (VMMs). Experimental results show a significant reduction in emulation time compared to conventional behavioral simulations. Additionally, an RRAM-based discrete Fourier transform (DFT) accelerator is analyzed as a case study featuring a range of in-depth system assessments.

## 1. Introduction

As technology advances and the demand for faster and more energy-efficient computing grows, the development of computational systems encounters significant challenges, particularly the memory-wall problem inherent in the conventional von Neumann architecture. The data movement between processing unit and memory becomes the main bottleneck limiting the system performance and energy efficiency [1]. This issue becomes critical today because modern applications, especially for artificial intelligence (AI) and other data-intensive computing, require massive amounts of data to be processed quickly and efficiently.

To address this issue, in-memory computing (IMC) systems are proposed as a promising computing paradigm. Unlike traditional computing architectures that rely on the constant data movement between memory and processing units, IMC directly embeds the computations within the memory itself, which significantly reduces the costly data transfer overhead [2].

Resistive RAM (RRAM) is an emerging memory technology that offers several benefits including high storage density, low energy consumption, multilevel storage, and rapid read operations [3]. Additionally, as a non-volatile memory (NVM), RRAM retains data without requiring a power supply, enhancing overall system efficiency. Due to the CMOS compatibility, the RRAM devices can be tightly integrated with the peripheral circuitry for write and read operations, resulting in promising read energy efficiency and latency [4].

In addition to the usage as memory for data storage, RRAM devices can be configured as crossbar arrays to implement IMC archietcures for

* Corresponding author.
*E-mail address:* wen@ihp-microelectronics.com (J. Wen).

https://doi.org/10.1016/j.microrel.2025.115630
Received 21 July 2024; Received in revised form 27 January 2025; Accepted 19 February 2025
Available online 12 March 2025

computing vector-matrix multiplications (VMMs). The matrix elements are stored in the RRAM crossbars as device conductance, while the vectors are encoded as input voltages applied on the crossbars. This architecture allows the VMMs to be performed in the analog domain with high parallelism and minimized data movements, which contributes to achieving highly efficient computations [5]. As a result, RRAM crossbars are highly attractive for implementing systems that require massive VMMs, such as hardware accelerators for artificial neural networks (ANNs) [6,7], homomorphic encryption (HE) [8], and Fourier transform [9]. A comprehensive review of hardware accelerators based on emerging technologies can be found in [10].

Although the RRAM technology exhibits promising potential, the corresponding design flow is not yet mature enough to enable rapid prototyping for holistic design space exploration [11]. One of the key challenges is developing device models that precisely capture device dynamics for simulations under various stimuli to accurately predict system behavior and evaluate hardware overhead [12]. Meanwhile, the device models need to be compact to ensure acceptable simulation time, especially as systems scale up to include millions of devices processing information in parallel to handle complex computational tasks [7].

The switching mechanism of RRAM devices relies on the growth and rapture of conductive filaments (CFs) in the oxide layer. The inherent stochastic nature leads to variability both between different devices and across multiple switching cycles, which are known as device-to-device and cycle-to-cycle variability, respectively [13]. Besides, some other non-ideal effects can affect the system reliability, such as conductance drift [14], read disturb [15], conductance fluctuation [16], and temperature dependency [17]. These non-ideal effects pose challenges in ensuring consistent performance and reliability across different devices and operating conditions. Furthermore, immature fabrication process can introduce defects that cause devices to become stuck in specific resistance states, leading to system malfunctions [18]. Therefore, it is crucial to model and manage these non-ideal effects throughout the hardware life cycle, starting from the system design stage [19].

To enable rapid simulations and expedite the iterative design optimizations, we introduced a field-programmable gate array (FPGA) based emulator for the RRAM crossbar array including peripheral circuitry in [20]. The emulator leverages a system-level behavioral model of the RRAM crossbar array described at the register-transfer level (RTL) [21]. This approach offers significant advantages over traditional central processing unit (CPU) based simulations. The contributions in [20] can be summarized as:

1. **Hardware Acceleration**: The RRAM crossbar model including peripheral circuitry is implemented on the FPGA platform, which outperforms the CPU-based behavioral simulations and achieves real-time emulations.
2. **Data-driven Variability Modeling**: A novel data-driven variability modeling approach based on multivariate kernel density estimation (KDE) is proposed. This approach effectively augments existing measurement data, extending data variety and allowing for the emulation of large-scale systems without data point reuse or additional experiments.
3. **Emulating Mapping and VMMs**: Leveraging the augmented data that models variability and incorporating modules like the write-verify algorithm, our platform can emulate system behaviors with variability for matrix mapping and VMM operations. This enhanced capability allows for more insightful analysis of system performance under realistic variability conditions.
4. **Energy Consumption Evaluation**: To assess the emulated hardware's efficiency, the emulator is interfaced with the NeuroSim framework [22]. This integration features the estimation of energy consumption for different operations.

In this paper, we extend our previous work in [20] with the following aspects:

1. **Expanded Device Characterization**: To rigorously validate the efficacy of the proposed data-driven augmentation approach for modeling device-to-device variability, the experimental dataset is substantially expanded. The number of characterized devices is increased from 256 to 4096, encompassing the entire memory array. This comprehensive characterization ensures a more accurate representation of the inherent variability within the device population.
2. **RRAM-based DFT Case Study**: A thorough analysis is conducted on an RRAM-based discrete Fourier transform (DFT) accelerator to investigate the trade-off between computational accuracy and energy consumption. This case study serves as a practical demonstration of the emulator's capability to assess system-level performance under realistic device variability conditions.

The reminder of this paper is organized as follows: Section 2 provides an overview of related works focusing on FPGA emulation for emerging technologies. Section 3 details the methodologies employed in this work, including LUT-based device modeling, data-driven variability modeling, and the architecture of the proposed hardware emulation platform. Section 4 presents a comprehensive evaluation of the FPGA emulator, and Section 5 introduces a case study demonstrating its efficacy. Finally, Section 6 summarizes the key findings and contributions of this work.

## 2. Related works

One of the challenges in simulating large-scale RRAM-based systems is the incorporation of complex device models that capture the inherent non-linear current–voltage (IV) characteristics and switching dynamics of individual devices. For instance, in certain models, the evolution of device conductance during resistive switching is attributed to the modulation of the gap between the CF and bottom electrode (BE) in the oxide layer, a process described by computationally intensive differential and hyperbolic equations [12]. This complexity is further compounded when considering large-scale systems with numerous devices operating in parallel, as the computational demands increase significantly. Additionally, incorporating device-to-device variability into the simulation model introduces another layer of complexity, further exacerbating the computational burden and simulation time.

FPGAs offer a promising platform for device-aware simulations of RRAM systems, effectively addressing the extensive computational demands posed by the inherent complex device and variability modeling. This enables efficient emulation of system behaviors, accelerating the development and optimization of RRAM-based systems. Leveraging their highly parallel processing capabilities, FPGAs facilitate high-speed emulation, overcoming the speed and scalability limitations often encountered in software-based simulations. This high-speed emulation becomes essential for large-scale RRAM systems. It also enables integrating the RRAM systems into multi-core processors like RISC-V as hardware accelerators [29] and co-simulations in real time. Moreover, their reconfigurability and flexibility make FPGAs well-suited for rapid prototyping and extensive design space exploration in the early stage of system development.

Several studies utilized FPGA emulators to accelerate simulations of systems based on emerging memory technologies, as summarized in Table 1. Ntinas et al. [23] implemented a 1D RRAM crossbar on FPGA, using a behavioral model of bipolar RRAM devices to emulate vector-vector multiplications in a single-layer ANN. This approach captured timing information crucial for cycle-accurate simulations, accurately reflecting device behavior at each time step. In contrast, Tolba et al. [24] emulated an RRAM-based multi-bit XNOR gate as the fundamental building block for an ANN, employing a simplified device model with five discrete conductance levels. While not explicitly mentioned, the omission of timing information in their model may have implications for achieving cycle-accurate simulations. Both studies [23,24] implemented pinched hysteresis loops for voltage-controlled resistance in

**Table 1**
Comparison of the proposed work against state-of-the-art methods.

| Work | [23] | [24] | [25] | [26] | [27] | [28] | This work |
|---|---|---|---|---|---|---|---|
| Topology | 1D Crossbar | XNOR | Crossbar | Crossbar | Register | Cache | Crossbar |
| Application | IMC | IMC | IMC | IMC | Storage | Storage | IMC |
| Scale | Small | Small | Large | Large | Large | Large | Large |
| Cycle-accurate | ✓ | N | × | × | ✓ | ✓ | ✓ |
| Device Programming | ✓ | × | × | × | ✓ | ✓ | ✓ |
| Device Variability | × | × | ✓ | ✓ | × | × | ✓ |
| Fault Injection | × | × | ✓ | ✓ | × | × | × |
| Energy Evaluation | × | × | × | × | ✓(Only dev.) | × | ✓ |

✓: Supported, ×: Not Supported, N: Not Given.

RRAM devices and compared hardware simulation results with the mathematical device models. However, a notable limitation in both works is the absence of device variability considerations, hindering insights into the impact of variability on system reliability.

Luo et al. [25] presented an emulator implemented on the FPGA platform for RRAM-based spiking neural networks (SNNs). Their design includes controllers and crossbar interconnections, facilitating comprehensive system-level evaluations. The authors explored the impact of network-on-chip (NoC) configurations on system performance through various core mapping schemes and buffer depths. Additionally, the integration of a noise generator enabled the study of hardware vulnerability to variability and stuck-at faults. However, this design prioritized scalability over cycle-level accuracy by serializing result accumulations within the crossbars. In [26], a FPGA-based emulation framework for RRAM-based ANNs was reported. This work offloaded massive VMMs in the crossbars to onboard multiply-and-accumulate (MAC) tree units for high parallelism. A runtime software stack with instructions and scheduling was implemented for flexible emulation of various ANN models. Variability and defects were injected by preprocessing weights streamed into MAC units. Both emulators [25,26] scarified the cycle-accurate emulation to achieve high performance and scalability. They focused on the inference phase to evaluate accuracy under device variations and faults, and the device programming process was simplified as standard memory write operations. Furthermore, none of the discussed works [23–26] considered energy efficiency, which is a crucial evaluation criterion for emerging technology-based IMC systems.

Beyond IMC systems, FPGA emulation has also been applied to systems utilizing emerging technologies as NVMs. Ruffini et al. [27] demonstrated an FPGA-based framework for NVM-based energy-harvesting intermittent computing systems, incorporating intentional delays in the read/write process to simulate latency for various operations. The authors implemented and tested different backup policies and compared the energy efficiency. However, their analysis focused solely on the energy consumption of the NVM devices themselves, neglecting the peripheral circuitry required to drive and interface with the devices. Zhao et al. [28] presented an FPGA-based framework for emulating various emerging NVM technologies as L1 data caches within a RISC-V processor environment. They evaluated the performance impact of different NVM configurations using benchmark applications, demonstrating the framework's capability to assess the performance of emerging memory technologies in realistic processor settings. Similar to [27], the device variations are not taken into account, which may impact the performance and reliability of the systems based on NVMs.

It is evident that existing emulators targeting large-scale IMC systems with emerging devices do not support cycle-accurate emulations, limiting detailed timing analysis. Furthermore, the device programming process, crucial in edge learning systems where frequent retraining and weight updates occur [30], is not adequately emulated in existing platforms. To validate and assess accelerators in realistic environments, integrating the system into a larger platform for simulation is necessary [28]. In such scenarios, real-time, cycle-accurate emulation becomes essential, enabling co-simulation with architectures like RISC-V for comprehensive system-level evaluations [29].

To address these gaps, we propose RRAMulator, an FPGA-based RRAM crossbar emulator. RRAMulator employs efficient device and variability modeling approaches, accurately capturing device behavior in large-scale systems without compromising emulation speed. Device variability is considered for programming and VMM operations with RRAM crossbars. By implementing behavioral models of peripheral circuitry and the write-verify algorithm interfacing with crossbars, RRAMulator can emulate the entire system's behavior under various scenarios. Additionally, energy efficiency, a key advantage of emerging technology-based IMC systems, is evaluated by feeding traces generated by emulator into NeuroSim framework [22]. Therefore, RRAMulator can emulate the system behaviors in a cycle-accurate manner, and evaluate the performance and energy efficiency under device variations. Notably, the proposed methodologies are not limited to RRAM-based systems and can be extended to other emerging technologies.

## 3. Methodology

### 3.1. LUT-based device modeling

Accurate device models are crucial for understanding the behavior and performance of emerging memory technologies like RRAM and guiding device engineering efforts. However, it is important to recognize a key distinction between device-level and system-level simulations. Device-level simulations focus on the intricate physical processes within individual devices, often requiring detailed models that capture the underlying physics. In contrast, system-level simulations aim to evaluate the performance of the entire memory system, including interactions between numerous devices and peripheral circuitry.

While detailed device models are essential for understanding device-level behavior, they can be computationally expensive and may not be strictly necessary for system-level simulations. System-level simulations prioritize capturing the essential device characteristics that significantly impact overall simulation speed. Therefore, deploying device models for system-level simulations needs to compromise between accuracy and computational complexity.

To minimize the computational overhead of device models and improve system simulation speed, we employ a look-up table (LUT) based approach [21]. This method focuses on capturing the essential IV relationship of the device, bypassing the need to solve complex equations that reflect device dynamics [12]. Similarly, another LUT-based device model is utilized in [31] to accelerate the analog simulations of the RRAM-based systems.

We demonstrate the efficacy of our proposed approaches using one-transistor-one-resistor (1T1R) RRAM devices manufactured at IHP with 130 nm SiGe BiCMOS technology. A memory chip containing 64 × 64 1T1R devices is characterized to collect data, which is subsequently used for analysis and evaluation of the proposed methods. This specific device serves as a representative example to showcase the adaptability of our modeling approaches across different device types and programming techniques.

Fig. 1 presents the schematic and corresponding transmission electron microscopy (TEM) image of an IHP 1T1R device. The metal–insulator–metal (MIM) stack, comprised of TiN/$HfO_2$/Ti/TiN, is monolithically fabricated with conventional CMOS in the back-end-of-line (BEOL) process. As a memory, information can be encoded as device conductance by applying voltages to the device terminals. Positive

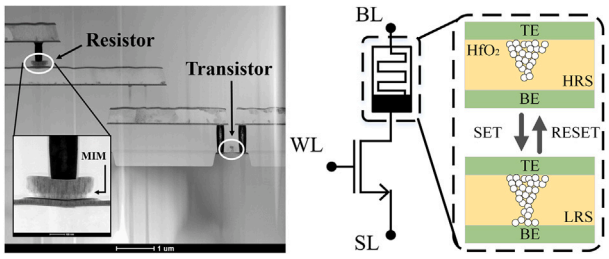



**Fig. 1.** Cross-sectional TEM image (left) [12] and schematic (center) of the 1T1R device. Resistive switching process with the change of CFs in the oxide layer (right) [20].

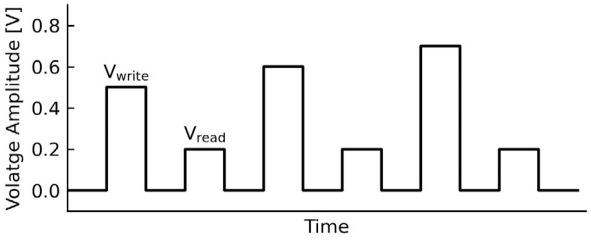



**Fig. 2.** Waveform of the ISPVA. Each write pulse with $V_{write}$ is followed by a read pulse with $V_{read}$ to check if the read out current reaches the predefined target with $I_{th}$. If not, a subsequent write pulse with the incremental amplitude is applied [13].

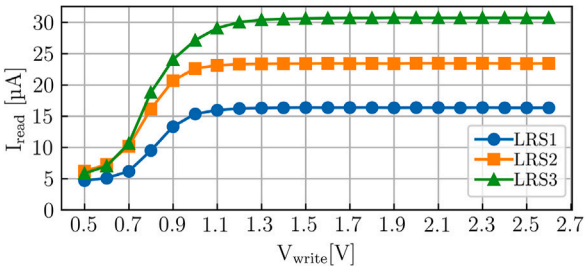



**Fig. 3.** The measured evolution of $I_{read}$ from HRS to three different LRSs. The current values are averaged over 4096 1T1R devices. $I_{read}$ are measured at a voltage of $V_{read} = 0.2$ V.

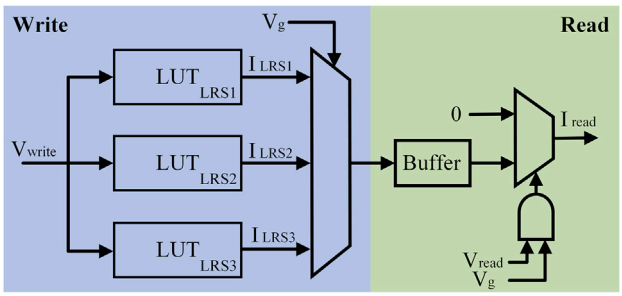



**Fig. 4.** Architecture of the LUT-based device modeling approach with write and read operations.

voltages between the bitline (BL) and sourceline (SL) switch the device from high resistance state (HRS) to low resistance state (LRS), while inverse polarity pulses reverse this transition. These processes are denoted as SET and RESET, respectively. The resistive switching behavior observed in RRAM devices is widely attributed to the formation and rupture of CFs within the oxide layer [12] , as depicted in Fig. 1. The NMOS transistor in the 1T1R device functions as a selector, enabling individual device access within a crossbar array and eliminating sneak path currents. Some device models describe the change of conductance during SET and RESET operations as a complex function of the gap between the CF and BE in the oxide layer, often involving differential and hyperbolic equations [12]. This complexity hinders their use in large-scale system simulations. Our LUT-based approach addresses this issue by capturing essential device behavior without the need for computationally expensive calculations.

To precisely control device conductance during programming and mitigate device-to-device variability, write-verify algorithms are commonly employed using pulse trains [7,13,32]. These write-verify algorithms consist of alternating write pulses and low-amplitude read pulses. The read pulse verifies whether the target conductance level has been achieved without disturbing the stored information. The write-verify algorithm should be implemented in the emulator as it directly impacts data fidelity and write latency. Accurately modeling this process ensures simulated behavior aligns with actual hardware, enabling effective algorithm evaluation and parameter optimization to achieve the optimal programming accuracy and speed. Additionally, it allows for a more realistic assessment of overall system performance and energy consumption, which are influenced by the write process.

In this work, we integrate the incremental step pulse with verify algorithm (ISPVA) into our emulator to achieve reliable multilevel storage [13]. Fig. 2 illustrates the voltage waveform deployed in this approach. After each write pulse with the amplitude of $V_{write}$ to alter the device conductance, a read pulse with $V_{read}$ is applied to determine if the output current reaches the predefined threshold ($I_{th}$). If not, another write pulse with higher amplitude is applied, incrementally increasing in fixed steps until $I_{th}$ is reached, indicating the completion of programming for the selected device.

During the SET operation, modulating the NMOS's gate voltage ($V_g$) applied to wordline (WL) controls compliance current flowing through the MIM stack, enabling transitions from HRS to various LRSs [13]. Reliable multilevel storage is realized by identifying appropriate pairs of $\{V_g, I_{th}\}$ for each resistance state based on device characterization. Fig. 3 shows the evolution of the mean readout currents ($I_{read}$) measured at a read voltage ($V_{read}$) of 0.2 V across 4096 devices, fabricated in a 1T1R memory array. The devices are programmed with ISPVA with an incremental voltage step of 0.1 V. The measured data demonstrates reliable storage of four distinct resistance states: one HRS and three LRSs.

To efficiently model large-scale systems with numerous devices operating in parallel, we employ a LUT-based device modeling approach. This method prioritizes modeling speed by focusing solely on the critical information for system-level simulations: the IV relationship. Instead of capturing detailed information like conductance changes resulting from voltage-induced variations in CF dimensions [12], the LUT-based approach provides a streamlined representation of device behavior.

The architecture of the LUT-based model, depicted in Fig. 4, comprises three LUTs, a buffer for each device, and two multiplexers. The LUTs store pre-generated IV relationship values for each LRS transiting from HRS. During write operations, the LUTs are swept to find the matched output currents corresponding to the input voltage ($V_{write}$) for each LRS. The matched output current for the target state is selected based on the simultaneously applied $V_g$ on the WL and stored in the buffer. For the FPGA implementation, the buffers can be realized with block RAMs (BRAMs), and LUTs can be shared among devices in crossbar arrays for efficiency. During read operations, if both $V_{read}$ and $V_g$ for the device are applied concurrently, indicating device is selected, the output current ($I_{read}$) is presented. Otherwise, the output remains zero. During VMMs operations, multiple devices sharing the same SL are selected simultaneously, and their outputs are accumulated.

The LUT-based modeling approach offers a lightweight, efficient method for device modeling, significantly accelerating system-level simulations. It captures device dynamics after each applied pulse, with model accuracy directly tied to the stored LUT values. This approach is highly flexible, allowing for the utilization of data samples generated from device-level models or directly measured experimental

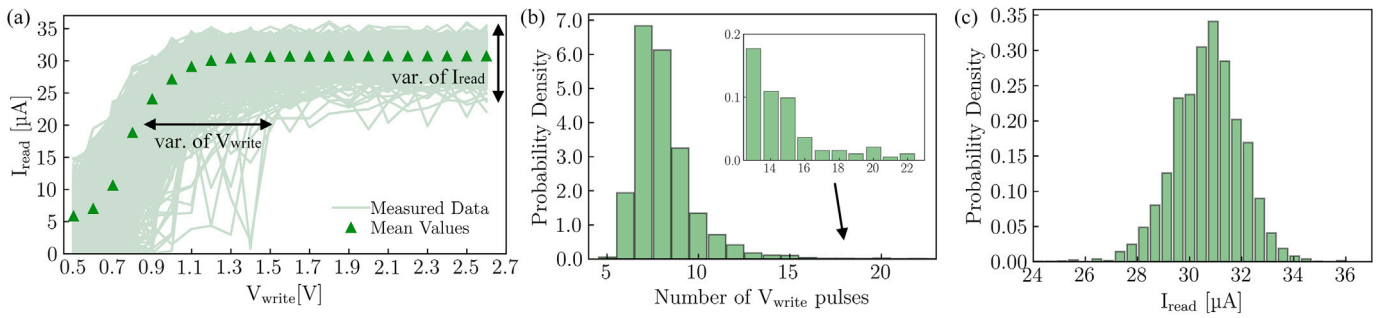



**Fig. 5.** Device-to-device variability in RRAM programming. (a) Evolution of $I_{read}$ for 4095 devices programmed to LRS3 using ISPVA. (b) Distribution of programming pulse counts, influencing latency and energy consumption for device programming. (c) Distribution of $I_{read}$ at the end of programming, impacting accuracy of the stored information and computations.

results. Furthermore, system-level variability and fault injection can be readily implemented by modifying the LUT values. By populating the LUTs with data from various emerging memory technologies and the corresponding writing algorithms, the system-level behavior can be investigated across a broad collections of device characteristics. This enables users to evaluate the performance implications of integrating different memory technologies into their systems.

While this model is limited to representing device behavior under the specific conditions of the simulation or measurement campaign used to generate the LUTs, this limitation can be mitigated by generating LUTs for a wide range of operating conditions or by employing adaptive techniques to update LUTs during simulation based on observed device behavior.

### 3.2. Device variability modeling

Device-to-device variability in RRAM technology arises from a combination of factors. Fluctuations in the oxide layer's thickness and composition during fabrication contribute to the observed differences in device behavior [33]. Additionally, during programming, the stochastic nature of CF formation leads to variations in their size and shape. Variability can significantly degrade the accuracy of stored information and computational results in RRAM-based systems. Device-to-device variability not only affects accuracy but also contributes to variations in the number of pulses required for state transitions, impacting programming latency and energy consumption. Therefore, incorporating this characteristic into system-level simulations is crucial for quantifying its impact and developing effective mitigation strategies. It is also necessary to accurately capture this information in system-level emulators without introducing significant computational overhead.

Fig. 5(a) illustrates the measured currents of devices programmed from HRS to LRS3, which is the LRS with the highest conductance. During programming, multilevel storage and variability reduction are achieved through the use of ISPVA.

All devices are initially subjected to a write voltage of 0.5 V. A read pulse of 0.2 V is then applied to each device, and the resulting read current is compared to a threshold current, which is 32 µA for LRS3. Devices that remain unswitched receive subsequent write pulses with incrementally increased amplitude of 0.1 V, each followed by a read pulse. For devices that have already switched, only the read pulse is applied to record their conductance. In summary, during programming, write pulses are applied exclusively to devices that have not yet reached the target state, while read pulses are applied to the entire array. The programming phase concludes once all devices achieve the threshold current. Note that one device failed to switch despite increased write pulses, hence only 4095 devices are shown in the plot.

During programming, two types of variability are relevant to system performance and reliability:

1. **Variability in Write Pulse Count**: The number of write pulses required to reach the target conductance level varies significantly across devices. As shown in Fig. 5(b), while most devices reach LRS3 within six to ten pulses, a subset exhibits tail distribution behavior, requiring upwards of 12 pulses. Ignoring or simplifying this tail distribution can lead to inaccurate system-level models and unrealistic performance predictions.
2. **Variability in Read Currents**: The variability in read currents after switching, as illustrated in Fig. 5(c), directly impacts stored information and computational accuracy. Overlapping read current distributions between different resistance states can introduce ambiguity and errors in data interpretation, potentially leading to incorrect computational results.

To accurately model device behavior and integrate it into system-level simulations with cycle accuracy, it is essential to simultaneously capture the two correlated device-to-device variability observed during programming. These variations impact both the latency and energy consumption to program devices in crossbars, as well as computational accuracy. The objective is to reproduce device behaviors illustrated in Fig. 5(a) while preserving the distributions of programming pulse counts and readout currents shown in (b) and (c).

While some device models incorporate simulations of device variability, incorporating such detailed models into system-level simulations often proves impractical due to scalability limitations and high computational costs. To overcome these challenges, data-driven approaches that leverage measured device data emerge as a more efficient alternative. One such approach, known as oversampling, involves randomly selecting devices from the existing measured dataset until the desired population size for the simulation is reached [15]. However, this method has limitations: when the target population exceeds the original dataset size, data points must be reused, preventing extrapolation beyond the measured data and limiting sample diversity. Another approach is fitting probability density functions (PDFs) to measured data for each write pulse, then generating new data samples for simulation [29]. It expands the data volume without directly reusing raw data. However, this approach also presents several challenges: first, fitting complex distributions observed in real devices to known PDFs might not always be feasible, and some approximations are required. Second, generating massive new data samples on-the-fly during runtime can be computationally intensive. Most importantly, the correlation between the two variability types may not be preserved since data generation for each device after each pulse is performed independently. This lack of captured correlation can lead to inaccurate system behavior representation, potentially yielding inaccurate performance predictions.

To overcome the limitations of the aforementioned data-driven approaches, we propose leveraging multivariate KDE method to extend the available data based on device measurements for variability modeling. Multivariate KDE, a non-parametric statistical method, estimates

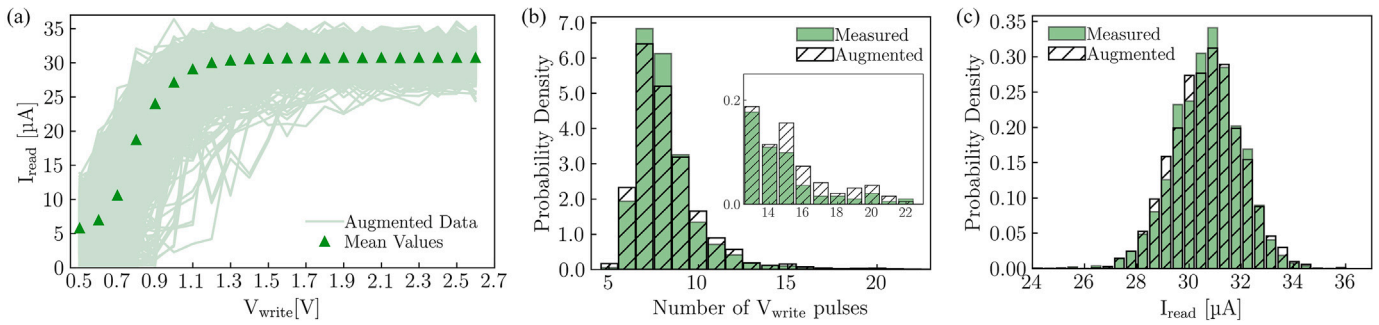



**Fig. 6.** The augmented data for RRAM device programming based on measured data. (a) The evolution of augmented $I_{read}$ for 4095 devices programmed to LRS3. (b) Comparison of the number of pulses required to program devices across augmented and raw data. (c) Comparison of raw and augmented data distributions for final $I_{read}$ values after programming.

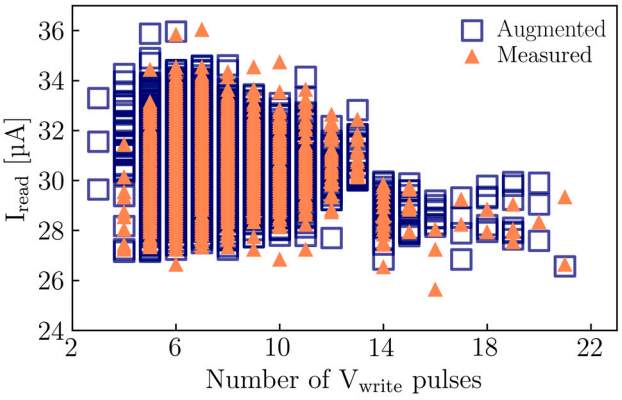



**Fig. 7.** The correlation between the number of write pulses and read currents for the raw and augmented data.

the PDF of multidimensional data without assuming the underlying parametric distribution [34]. Unlike oversampling, which relies on the repetition of existing data points, our KDE-based approach generates new data points that capture the underlying statistical characteristics of the original measurements. In our case, the multidimensional data consists of current evolution traces measured across devices during programming cycles shown in Fig. 5(a). By employing Gaussian kernel functions within KDE, we construct a smooth, continuous, and joint PDF that accurately captures the complex relationships and correlations between different aspects of device behavior. This refined PDF serves as a generative model, enabling us to synthesize new current evolution traces that exhibit the same statistical properties as the original data. In other words, the generated traces are statistically close to the measured ones, but they provide additional samples to expand the dataset and improve the sample diversity in our system-level simulations.

Fig. 6(a) shows the augmented current evolutions of devices programmed from HRS to LRS3, generated based on measurements of real devices shown in Fig. 5. For direct comparison, the number of synthesized traces matches the original dataset size. However, the data generation is not limited to this quantity, and a significantly larger number of data samples can be created. Comparing the distributions of programming duration and final current values in Fig. 6(b) and (c) reveals a close alignment between the generated data and the raw data traces, validating the effectiveness of the proposed data augmentation method. This leads to the generation of realistic and diverse synthetic data traces that faithfully reflect the characteristics of the real devices.

To further validate the preservation of correlations between write pulse count and read current variability in the generated data, we plot the correlation between these two variables in Fig. 7. The strong agreement between the raw and augmented data distributions confirms that the synthesized data accurately captures the inherent correlations present in the original measurements. Both distributions show similar trends: devices requiring less write pulses tend to have wider distributions in their final $I_{read}$ values. Besides, the devices requiring more number of pulses to switch lead to a low-amplitude $I_{read}$ after switching, which implies the devices that are more difficult to switch may ultimately settle into lower conductance. Maintaining this correlation is essential for ensuring the accuracy of system-level simulations.

Multivariate KDE provides a valuable tool for augmenting initial RRAM measurement data, expanding the available dataset for accurate modeling of large-scale RRAM systems with numerous devices without the need to reuse raw data. By pre-generating data traces through multivariate KDE, this approach can be seamlessly integrated with the LUT-based device modeling methodology. By combining these two approaches, we can incorporate comprehensive variability modeling into our FPGA emulator, accommodating a large number of devices with diverse behavior characteristics. This allows us to accurately simulate the impact of device-to-device variability on RRAM crossbar performance, including metrics such as programming latency, energy consumption, and computational accuracy. The detailed implementation of this combined approach on the FPGA platform is presented in the following subsection.

### 3.3. System architecture of RRAMulator

FPGA technology, with its inherent reconfigurability and high parallelism, offers a promising platform for cycle-accurate emulation of large-scale RRAM systems, effectively capturing the impact of device-to-device variability.

Fig. 8 shows the overall architecture of the proposed FPGA-based RRAM crossbar emulator. RRAMulator implements essential operations for RRAM crossbar arrays, including write, read for memory applications, and VMMs for IMC applications. To facilitate efficient data transfer between the PC and the FPGA emulator, we developed a C-based software program using the Xilinx Vivado software development kit (SDK). This program manages the transmission of input frames to the emulator and retrieves the processed data from the BRAMs. Input frames contain predefined operational codes specifying the desired operation (e.g., programming, read, VMM), along with the addresses of the target RRAM devices within the crossbar array. These frames also include the necessary data payloads, such as matrix values for programming or input vectors for VMM operation. To minimize communication overhead, burst mode transmission is supported, allowing for the efficient transfer of input and output data for an entire column or row with a single operation. Input frames are transmitted to the programmable logic (PL) of the FPGA through general-purpose input/output (GPIO) interfaces and are subsequently processed by an instruction decoder. The decoder then parses the input frames, extracting relevant instructions and data, which are then forwarded to the RRAM crossbar array for execution.

The RRAM crossbar, depicted in Fig. 8(b), comprises the emulated RRAM devices and peripheral circuitry. Some peripheral components

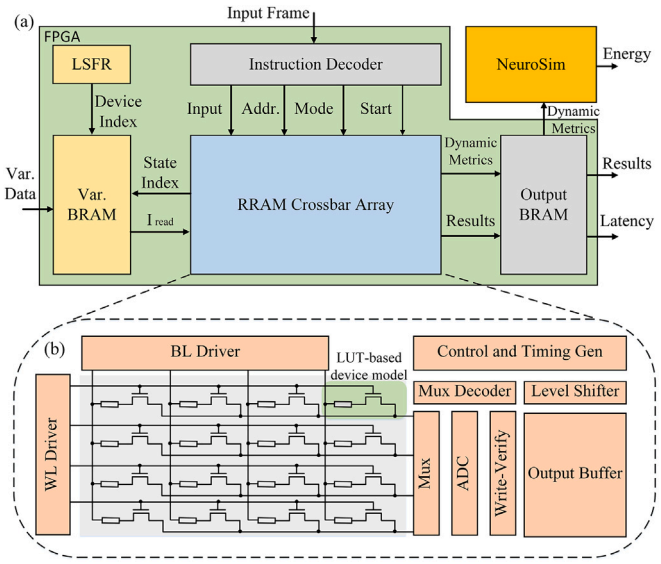



**Fig. 8.** (a) System architecture of the proposed FPGA-based RRAM emulator. (b) Detailed architecture of the emulated RRAM crossbar array including peripheral circuitry [20].

essential for RRAM operation are mixed-signal circuits that cannot be fully emulated on an FPGA. To ensure cycle-accurate emulation, we construct behavioral models of these components, incorporating crucial timing information to accurately replicate their functionality.

When an operation is initiated, the control unit orchestrates the system by dispatching control signals to the BL and WL drivers, adhering to the generated timing information defining pulse width, amplitude, and synchronization. This synchronization ensures that the voltages applied on the BL and WL coincide precisely during the programming and read operations, mimicking the behavior of pulses driving the RRAM devices. Furthermore, the amplitudes of the BL and WL voltages determine the conductance value to be stored in the device model of the selected cell. At the output, successive approximation register (SAR) based analog-to-digital converters (ADCs) are shared across rows via a multiplexer to amortize the high hardware costs [2]. To accurately model SAR ADC latency, the ADC output is streamed out with shift registers.

Device-to-device variability is represented by pre-loading augmented data traces, derived for each resistance state via multivariate KDE, into a dedicated variability BRAM. A linear-feedback shift register (LSFR) generates uniformly distributed random numbers to sample these traces during runtime. When a device programming operation is initiated, a random number is recorded as an index, along with the target resistance state's index. This combined index fetches the corresponding data trace from the variability BRAM, which dictates the specific LUT used in the device model to represent the device's current evolution during programming. Incorporating this trace allows the emulated device model to exhibit realistic variations in programming pulse count and read currents, reflecting the inherent statistical variability of real RRAM devices. This implementation is advantageous as the computationally intensive data augmentation with multivariate KDE is performed offline, leaving only the negligible latency of data retrieval from the variability BRAM as overhead during emulation. This ensures that variability modeling does not bottleneck the overall emulation speed. In addition to device-to-device variability, cycle-to-cycle variability can also be integrated into the platform by storing cycle-specific data in the variability BRAM along with a corresponding cycle index. During programming, variability data can be accessed based on both device and cycle indices, enabling accurate modeling of cycle-to-cycle and device-to-device variations simultaneously.

The ISPVA write-verify algorithm is implemented within the emulator. Threshold values for different conductance states are configured via the developed software. The pulse trains comprised of $V_{write}$ and $V_{read}$ in Fig. 2 are generated automatically by the control unit with BL and WL drivers.

During programming, an iterative process takes place between the ADC outputs, the fetched data trace, and the control unit. The write-verify module compares the ADC outputs to the configured thresholds. If the ADC outputs fall below the threshold for the target conductance state, it indicates that the device has not yet transitioned to the target state. In this case, the control unit generates the next $V_{write}$ with the incremental amplitude. The device model retrieves the next data point from the fetched data trace, which continues the verification. Conversely, if the ADC outputs exceed the threshold, it signifies that the device has successfully transitioned to the target state. The control unit then terminates the programming sequence for that device and proceeds to the next one. This interactive interplay ensures that the write-verify algorithm adapts the programming pulse sequence based on the real-time behavior of the emulated device, incorporating the inherent variability. For VMM operations, the write-verify algorithm is not employed. Instead, the digitized accumulation results are stored directly in the output BRAM and can be accessed via the implemented program.

The trace-based energy consumption for each operation is estimated using the modified NeuroSim framework [22], with technology-dependent parameters calibrated to the IHP 130 nm process design kit (PDK). Upon completion of an emulated operation, the framework receives some relevant dynamic metrics dumped from the emulator during that operation. For example, pulse counts and $I_{read}$ after each pulse from the programming phase are used to estimate programming energy, while input vectors and stored conductance values contribute to the energy consumption estimation for VMMs.

In summary, our FPGA-based RRAM crossbar emulator, RRAMulator, with its cycle-accurate emulation of RRAM crossbar behavior, incorporates a modified NeuroSim framework to estimate trace-based energy consumption. We provide a comprehensive platform for evaluating the energy efficiency and overall performance of RRAM-based systems, accounting for device variability and programming algorithms.

## 4. Evaluation

To evaluate the proposed RRAMulator framework, we conducted a series of experiments focusing on two key aspects: emulation speed compared to traditional CPU-based behavioral simulations, and energy analysis of an RRAM-based DFT accelerator. RRAMulator is developed in Verilog and implemented on a Xilinx ZCU102 evaluation board using the Vivado Design Suite. It features configurable parameters to enhance adaptability. In the experiments, pulse widths for $V_{write}$ and $V_{read}$ are set to 1 μs and 100 ns, respectively, aligning the 10 MHz system clock with the $V_{read}$ pulse width. The $V_{write}$ pulse width is chosen to match the device measurements with ISPVA [13].

To evaluate the emulation speed and highlight the advantages of RRAMulator, we map the designs with various crossbar sizes onto the FPGA, systematically increasing the scale to assess performance under different workloads. For each crossbar configuration, the input matrices consisting of 2-bit values are randomly generated with an equal distribution of states. We measure the time required to complete a set of representative operations, including programming the whole crossbar followed by 1000 times of VMMs. To quantify the speedup achieved by RRAMulator, we compare the emulation time against the corresponding simulation time obtained by running the same Verilog design on a conventional CPU using the Vivado simulator. This comparative analysis demonstrates the effectiveness of RRAMulator in accelerating RRAM system simulations.

A primary goal of RRAMulator is to achieve real-time emulation, where the emulation speed precisely matches real-world timing. In our

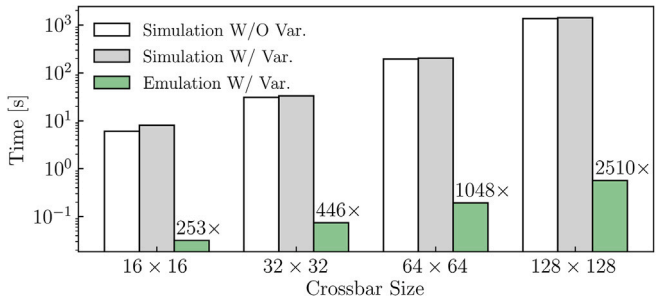



**Fig. 9.** Comparison between the behavioral simulation and emulation time for various sizes of crossbar arrays in logarithmic scale. The evaluation involves mapping a matrix containing 2-bit values to a crossbar followed by 1000 times VMM operations [20].

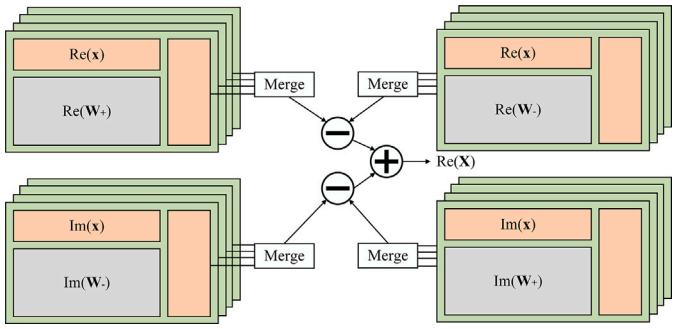



**Fig. 10.** Computing DFT with RRAM crossbars. Real and imaginary parts of twiddle factors are mapped as device conductance onto separate crossbars.

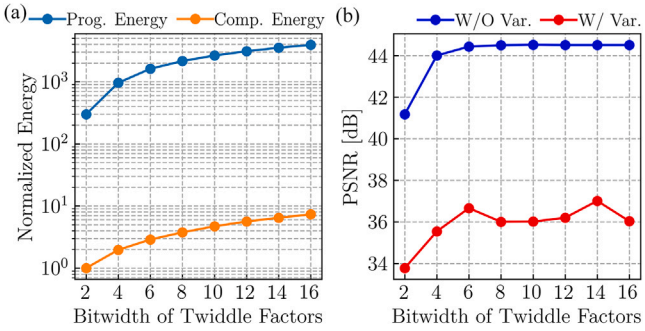



**Fig. 11.** (a) Comparison of normalized energy consumption for programming twiddle factors onto RRAM crossbars versus performing computations with different bitwidths for 128-point DFT. (b) PSNR of DFT computation results as a function of twiddle factor bitwidth.

case, the timing parameters for ISPVA are specifically adopted, as all device characterizations are obtained based on this algorithm. This is crucial for accurately interfacing with external high-performance platforms like RISC-V processors, which operate on real-world timescales. While the clock frequency of RRAMulator can be increased to accelerate the simulation, doing so would result in a design speed faster than real time, causing synchronization issues and mismatches with the external platform. Conversely, a slower emulation speed can introduce delays and bottlenecks in the overall system. Therefore, maintaining real-time emulation is essential for seamless integration and accurate interaction with external platforms.

The results in Fig. 9 show a significant speedup achieved by the RRAMulator compared to conventional CPU-based behavioral simulations. This speedup is particularly pronounced for larger crossbar sizes, highlighting the emulator's ability to efficiently handle the increasing complexity and parallelism inherent in such systems. The fact that CPU-based simulations are far from real time further emphasizes the advantage of the FPGA platform, which leverages its high parallelism to achieve fast emulation.

Furthermore, the inclusion of device-to-device variability in the model incurs minimal overhead, as evidenced by the marginal difference in simulation time between scenarios with and without the consideration of variability. This efficient variability modeling approach, coupled with the LUT-based device model, enables real-time emulation while accurately capturing the impact of device variations on system behavior.

Regarding resource utilization, the 128 × 128 RRAM crossbar array, including peripheral circuitry, consumes approximately 10% of the available LUTs, 7% of the BRAM, and 4% of the flip-flops on the Xilinx ZCU102 board. This demonstrates the model's potential for scalability to larger crossbar sizes while maintaining reasonable resource consumption.

In summary, the proposed FPGA-based emulator serves as a highly flexible and efficient platform for emulating RRAM-based systems, offering real-time performance that is not achievable with CPU-based simulations. Unlike application-specific integrated circuit (ASIC) implementations, which are optimized for speed and power efficiency in final applications, the FPGA emulator is designed to support rapid prototyping and system-level evaluation during the development phase. Compared to other emulation platforms, such as CPU and GPU-based frameworks, the FPGA implementation achieves significantly lower latency and allows hardware-in-the-loop co-simulations, making it particularly suited for cycle-accurate evaluations of device and system behaviors.

## 5. Case study

To showcase the capabilities of our proposed RRAMulator framework, we simulate an RRAM-based DFT accelerator. DFT is a fundamental algorithm in signal processing, image compression, and telecommunications, converting time-domain signals into their frequency-domain representations. This transformation is achieved through VMMs between input symbols and a matrix of complex-valued twiddle factors, which can be expressed as:

$$\mathbf{X} = \mathbf{W}\mathbf{x}. \tag{1}$$

Leveraging the inherent parallelism and energy efficiency of RRAM crossbars, the DFT can be performed in the analog domain by mapping the twiddle factors onto the crossbar as device conductance values. To accommodate the complex nature of twiddle factors, which comprise real and imaginary components, separate computations are conducted for each part.

Fig. 10 details the system architecture of the RRAM-based DFT accelerator, and it should be noted that only the computation of the real part of **X** is depicted. Signed computations are achieved by mapping twiddle factors as conductance difference between two crossbars [6]. To meet the resolution requirements of DFT computations, each twiddle factor is mapped to a cluster of devices, and bit significance is recovered through shift-and-add operations on the digitized outputs [35].

To evaluate energy consumption and computational accuracy, we simulate a 128-point DFT using RRAM crossbars with augmented device data to account for device-to-device variability. Twiddle factors are mapped to RRAM devices with 2-bit resolution using ISPVA and 8-bit ADCs for readout. The inputs are encoded as binary numbers with 6-bit resolution.

Fig. 11(a) demonstrates the energy consumption for programming twiddle factors onto the crossbars and computations relative to the bitwidth of coefficients. The results are normalized to the energy consumed at a minimal bitwidth of two, namely, each signed twiddle factor mapped to a differential device pair. It can be noticed that a significant gap exists between programming and computational energy. At the minimum bitwidth, programming consumes approximately 325 times more energy than computation. This disparity increases with bitwidth,

reaching a factor of 618 at a bitwidth of eight. The results highlight the importance of minimizing frequent device reprogramming to avoid the high energy cost associated with this process. To mitigate the impact of high programming energy, it is essential to perform a large number of computations per programming cycle. This amortizes the programming cost over multiple computations, enhancing the overall energy efficiency of RRAM-based DFT accelerators. Leveraging the non-volatility of RRAM devices, twiddle factors can be retained after power-off. However, protecting devices against dynamic effects such as read disturb and temperature dependency is crucial, as these non-ideal effects can distort stored values and necessitate energy-intensive reprogramming for data recovery.

To assess the computational accuracy of the system and the impact of device variability, we utilize the peak signal-to-noise ratio (PSNR) metric. PSNR is a widely used measure in signal processing that quantifies the quality of signals by comparing the maximum possible power of a signal to the power of corrupting noise. In our evaluation, we compute the PSNR of the signal processed by the RRAM-based DFT accelerator against a reference signal obtained from an ideal DFT implementation. This allows us to quantify the degree of signal degradation introduced by the RRAM device's inherent variability and the approximation errors in the analog computation. A higher PSNR value indicates better signal quality and closer agreement with the ideal DFT output. The PSNR can be computed as:

$$\mathrm{PSNR} = 10\log_{10}\left(\frac{\mathrm{MAX}^2}{\mathrm{MSE}}\right), \tag{2}$$

where MAX is the maximum value of the signal and MSE is the mean squared error between the RRAM-based DFT outputs and the reference.

Fig. 11(b) presents the PSNR of the RRAM-based DFT accelerator for various twiddle factor resolutions, both with and without considering device-to-device variability. Each configuration contains 1000 experiments with randomly generated input signals containing complex numbers. The results reveal a substantial margin between the PSNR with and without variability, indicating that device variability significantly impacts computational accuracy. Ignoring device-to-device variability in simulations may lead to overly optimistic accuracy estimates. Furthermore, the PSNR initially increases with higher twiddle factor resolutions in both scenarios but eventually plateaus, showing that beyond a certain resolution, factors such as ADC precision and input data resolution become the primary limitations to computational accuracy. The gap between the two curves suggests potential for further improvement in the accuracy of RRAM-based DFT accelerators by developing reliable mapping techniques to mitigate the impact of device variability, such as the variability-aware progressive device mapping scheme proposed in [9].

## 6. Conclusion

In this work, we present RRAMulator, an efficient FPGA-based emulation platform for RRAM-based IMC systems. To improve system performance, we employ a lightweight LUT-based device model that avoids the complexity of traditional device models in system-level simulations. We enhance this approach with a data-driven variability modeling technique based on multivariate KDE, which leverages device measurements to expand the available data and accurately model device-to-device variations. By combining these two approaches, we achieve real-time emulation of RRAM crossbars, enabling co-simulation with other designs for seamless integration. Furthermore, we implement the RRAM crossbar array with behavioral models of peripheral circuitry on the FPGA, accurately reflecting the behavior and latency of the system. We integrate the NeuroSim framework to estimate energy consumption based on traces generated during emulation, providing a comprehensive assessment of system efficiency. Our implementation on an FPGA demonstrates a significant speedup factor up to 2510 times compared to CPU-based simulations for a 128 × 128 crossbar, while maintaining low hardware resource utilization. The RRAM-based DFT simulation conducted on RRAMulator reveals insights into energy efficiency and computational accuracy, showcasing the platform's potential for deep analysis of the systems. Overall, RRAMulator offers an efficient and comprehensive framework for modeling and evaluating RRAM-based IMC systems, accelerating the system design space exploration and development.

## CRediT authorship contribution statement

**Jianan Wen:** Conceptualization, Methodology, Software, Formal analysis, Investigation, Visualization, Writing – original draft, Writing – review & editing. **Fabian Luis Vargas:** Methodology, Formal analysis, Validation, Writing – review & editing. **Fukun Zhu:** Investigation, Software, Visualization, Validation. **Daniel Reiser:** Conceptualization, Writing – review & editing. **Andrea Baroni:** Resources, Data curation, Writing – review & editing. **Markus Fritscher:** Conceptualization, Writing – review & editing. **Eduardo Perez:** Resources, Data curation, Writing – review & editing. **Marc Reichenbach:** Resources, Writing – review & editing. **Christian Wenger:** Project administration, Funding acquisition, Writing – review & editing. **Milos Krstic:** Supervision, Project administration, Funding acquisition, Writing – review & editing.

## Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

## Acknowledgment

The authors acknowledge the financial support by the Federal Ministry of Education and Research of Germany in the program of "Souverän. Digital. Vernetzt". Joint project 6G-RIC, project identification number: 16KISK026.

## Data availability

The data that has been used is confidential.

## References

[1] V. Sze, Y. Chen, T. Yang, J. Emer, Efficient processing of deep neural networks: A tutorial and survey, Proc. IEEE 105 (2017) 2295–2329.
[2] S. Yu, H. Jiang, S. Huang, X. Peng, A. Lu, Compute-in-memory chips for deep learning: Recent trends and prospects, IEEE Circuits Syst. Mag. 21 (2021) 31–56.
[3] H. Wong, H. Lee, S. Yu, Y. Chen, Y. Wu, P. Chen, B. Lee, F. Chen, M. Tsai, Metal–oxide RRAM, Proc. IEEE 100 (2012) 1951–1970.
[4] L. Upton, A. Levy, M. Scott, D. Rich, W. Khwa, Y. Chih, M. Chang, S. Mitra, P. Raina, B. Murmann, EMBER: A 100 MHz, 0.86 mm2, multiple-bits-per-cell RRAM macro in 40 nm CMOS with compact peripherals and 1.0 pj/bit read circuitry, in: ESSCIRC 2023- IEEE 49th European Solid State Circuits Conference, ESSCIRC, 2023, pp. 469–472.
[5] M. Hu, J. Strachan, Z. Li, E. Grafals, N. Davila, C. Graves, S. Lam, N. Ge, J. Yang, R. Williams, Dot-product engine for neuromorphic computing: Programming 1t1m crossbar to accelerate matrix–vector multiplication, in: 2016 53nd ACM/EDAC/IEEE Design Automation Conference, DAC, 2016, pp. 1–6.
[6] P. Yao, H. Wu, B. Gao, J. Tang, Q. Zhang, W. Zhang, J. Yang, H. Qian, Fully hardware-implemented memristor convolutional neural network, Nature 577 (2020) 641–646, http://dx.doi.org/10.1038/s41586-020-1942-4.
[7] W. Wan, R. Kubendran, C. Schaefer, S. Eryilmaz, W. Zhang, D. Wu, S. Deiss, P. Raina, H. Qian, B. Gao, S. Joshi, H. Wu, H. Wong, G. Cauwenberghs, A compute-in-memory chip based on resistive random-access memory, Nature 608 (2022) 504–512, http://dx.doi.org/10.1038/s41586-022-04992-8.
[8] X. Li, B. Gao, B. Lin, R. Yu, H. Zhao, Z. Wang, Q. Qin, J. Tang, Q. Zhang, X. Li, Z. Hao, X. Li, D. Kong, L. Ma, N. Deng, H. Qian, H. Wu, First demonstration of homomorphic encryption using multi-functional RRAM arrays with a novel noise-modulation scheme, in: 2022 International Electron Devices Meeting, IEDM, 2022, pp. 33.5.1–33.5.4.

[9] J. Wen, A. Baroni, E. Perez, M. Uhlmann, M. Fritscher, K. KrishneGowda, M. Ulbricht, C. Wenger, M. Krstic, Towards reliable and energy-efficient RRAM based discrete Fourier transform accelerator, in: 2024 Design, Automation and Test in Europe Conference and Exhibition, DATE, 2024, pp. 1–6.

[10] A. Mehonic, D. Ielmini, K. Roy, O. Mutlu, S. Kvatinsky, T. Serrano-Gotarredona, B. Linares-Barranco, S. Spiga, S. Savelev, A. Balanov, et al., Roadmap to neuromorphic computing with emerging technologies, 2024, arXiv preprint arXiv:2407.02353.

[11] S. Maheshwari, S. Stathopoulos, J. Wang, A. Serb, Y. Pan, A. Mifsud, L. Leene, J. Shen, C. Papavassiliou, T. Constandinou, T. Prodromakis, Design flow for hybrid CMOS/Memristor systems—Part I: Modeling and verification steps, IEEE Trans. Circuits Syst. I: Regul. Pap. 68 (2021) 4862–4875.

[12] E. Pérez-Bosch Quesada, R. Romero-Zaliz, E. Pérez, M. Kalishettyhalli Mahadevaiah, J. Reuben, M. Schubert, F. Jiménez-Molinos, J. Roldán, C. Wenger, Toward reliable compact modeling of multilevel 1T-1r RRAM devices for neuromorphic systems, Electron. 10 (2021) https://www.mdpi.com/2079-9292/10/6/645.

[13] E. Pérez, C. Zambelli, M. Mahadevaiah, P. Olivo, C. Wenger, Toward reliable multi-level operation in RRAM arrays: Improving post-algorithm stability and assessing endurance/data retention, IEEE J. Electron Devices Soc. 7 (2019) 740–747.

[14] D. Reiser, M. Reichenbach, T. Rizzi, A. Baroni, M. Fritscher, C. Wenger, C. Zambelli, D. Bertozzi, Technology-aware drift resilience analysis of RRAM crossbar array configurations, in: 2023 21st IEEE Interregional NEWCAS Conference, NEWCAS, 2023, pp. 1–5.

[15] J. Wen, A. Baroni, E. Perez, M. Ulbricht, C. Wenger, M. Krstic, Evaluating read disturb effect on RRAM based AI accelerator with multilevel states and input voltages, in: 2022 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems, DFT, 2022, pp. 1–6.

[16] E. Quesada, M. Mahadevaiah, T. Rizzi, J. Wen, M. Ulbricht, M. Krstic, C. Wenger, E. Perez, Experimental assessment of multilevel RRAM-based vector-matrix multiplication operations for in-memory computing, IEEE Trans. Electron. Devices 70 (2023) 2009–2014.

[17] T. Rizzi, A. Baroni, A. Glukhov, D. Bertozzi, C. Wenger, D. Ielmini, C. Zambelli, Process-voltage-temperature variations assessment in energy-aware resistive RAM-based FPGAs, IEEE Trans. Device Mater. Reliab. 23 (2023) 328–336.

[18] L. Poehls, M. Fieback, S. Hoffmann-Eifert, Al, Review of manufacturing process defects and their effects on memristive devices, J. Electron. Test. 37 (2021) 427–437.

[19] M. Fieback, L. Pöhls, Lifecycle management of emerging memories, in: 2024 IEEE European Test Symposium, ETS, 2024, pp. 1–6.

[20] J. Wen, F. Vargas, F. Zhu, D. Reiser, A. Baroni, M. Fritscher, E. Perez, M. Reichenbach, C. Wenger, M. Krstic, Cycle-accurate FPGA emulation of RRAM crossbar array: Efficient device and variability modeling with energy consumption assessment, in: 2024 IEEE 25th Latin American Test Symposium, LATS, 2024, pp. 1–6.

[21] J. Wen, M. Ulbricht, E. Perez, X. Fan, M. Krstic, Behavioral model of dot-product engine implemented with 1t1r memristor crossbar including assessment, in: 2021 24th International Symposium on Design and Diagnostics of Electronic Circuits and Systems, DDECS, 2021, pp. 29–32.

[22] X. Peng, S. Huang, Y. Luo, X. Sun, S. Yu, DNN+NeuroSim: An end-to-end benchmarking framework for compute-in-memory accelerators with versatile device technologies, in: 2019 IEEE International Electron Devices Meeting, IEDM, 2019, pp. 32.5.1–32.5.4.

[23] V. Ntinas, I. Vourkas, A. Abusleme, G. Sirakoulis, A. Rubio, Experimental study of artificial neural networks using a digital memristor simulator, IEEE Trans. Neural Netw. Learn. Syst. 29 (2018) 5098–5110.

[24] M. Tolba, W. Sayed, M. Fouda, H. Saleh, M. Al-Qutayri, B. Mohammad, A. Radwan, Digital emulation of a versatile memristor with speech encryption application, IEEE Access. 7 (2019) 174280–174297.

[25] T. Luo, X. Wang, C. Qu, M. Lee, W. Tang, W. Wong, R. Goh, An FPGA-based hardware emulator for neuromorphic chip with RRAM, IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. 39 (2020) 438–450.

[26] Y. Shi, Y. Sun, J. Jiang, G. He, Q. Wang, N. Jing, Fast FPGA-based emulation for reram-enabled deep neural network accelerator, in: 2021 IEEE International Symposium on Circuits and Systems, ISCAS, 2021, pp. 1–5.

[27] S. Ruffini, L. Caronti, K. Yıldırım, D. Brunelli, NORM: An FPGA-based non-volatile memory emulation framework for intermittent computing, J. Emerg. Technol. Comput. Syst. 18 (2022) http://dx.doi.org/10.1145/3517812.

[28] Y. Zhao, S. Ullah, S. Sahoo, A. Kumar, NvMISC: Toward an FPGA-based emulation platform for RISC-V and nonvolatile memories, IEEE Embed. Syst. Lett. 15 (2023) 170–173.

[29] M. Fritscher, A. Veronesi, A. Baroni, J. Wen, T. Spätling, M. Mahadevaiah, N. Herfurth, E. Perez, M. Ulbricht, M. Reichenbach, A. Hagelauer, M. Krstic, Prototyping reconfigurable RRAM-based AI accelerators using the RISC-v ecosystem and digital twins, High Perform. Comput. (2023) 500–514.

[30] W. Zhang, P. Yao, B. Gao, Q. Liu, D. Wu, Q. Zhang, Y. Li, Q. Qin, J. Li, Z. Zhu, Y. Cai, D. Wu, J. Tang, H. Qian, Y. Wang, H. Wu, Edge learning using a fully integrated neuro-inspired memristor chip, Sci. 381 (2023) 1205–1211.

[31] M. Uhlmann, T. Rizzi, J. Wen, E. Pérez-Bosch Quesada, B. Al Beattie, K. Ochs, E. Pérez, P. Ostrovskyy, C. Carta, C. Wenger, G. Kahmen, LUT-based RRAM model for neural accelerator circuit simulation, in: Proceedings of the 18th ACM International Symposium on Nanoscale Architectures, 2024.

[32] A. Baroni, A. Glukhov, E. Pérez, C. Wenger, D. Ielmini, P. Olivo, C. Zambelli, Low conductance state drift characterization and mitigation in resistive switching memories (RRAM) for artificial neural networks, IEEE Trans. Device Mater. Reliab. 22 (2022) 340–347.

[33] E. Brum, M. Fieback, T. Copetti, H. Jiayi, S. Hamdioui, F. Vargas, L. Poehls, Evaluating the impact of process variation on RRAMs, in: 2021 IEEE 22nd Latin American Test Symposium, LATS, 2021, pp. 1–6.

[34] J. Hwang, S. Lay, A. Lippman, Nonparametric multivariate density estimation: A comparative study, IEEE Trans. Signal Process. 42 (1994) 2795–2810.

[35] A. Shafiee, A. Nag, N. Muralimanohar, R. Balasubramonian, J. Strachan, M. Hu, R. Williams, V. Srikumar, ISAAC: a convolutional neural network accelerator with in-situ analog arithmetic in crossbars, in: Proceedings of the 43rd International Symposium on Computer Architecture, 2016, pp. 14–26, http://dx.doi.org/10.1109/ISCA.2016.12.