Integrated Architecture for Neural Networks and Security Primitives using RRAM Crossbar

Simranjeet Singh*¶§, Furqan Zahoor †§, Gokulnath Rajendran†, Vikas Rana¶,
Sachin Patkar*, Anupam Chattopadhyay†, Farhad Merchant‡
*Indian Institute of Technology, Bombay, †Nanyang Technological University, Singapore,
¶Forschungszentrum Jülich GmbH, ‡Newcastle University, UK
{simranjeet, patkar}@ee.iitb.ac.in, {furqan.zahoor@, gokulnat002@e., anupam@}ntu.edu.sg,
{si.singh, v.rana}@fz-juelich.de, farhad.merchant@newcastle.ac.uk

***Abstract*—This paper proposes an architecture that integrates neural networks (NNs) and hardware security modules using a single resistive random access memory (RRAM) crossbar. The proposed architecture enables using a single crossbar to implement NN, true random number generator (TRNG), and physical unclonable function (PUF) applications while exploiting the multi-state storage characteristic of the RRAM crossbar for the vector-matrix multiplication operation required for the implementation of NN. The TRNG is implemented by utilizing the crossbar's variation in device switching thresholds to generate random bits. The PUF is implemented using the same crossbar initialized as an entropy source for the TRNG. Additionally, the weights locking concept is introduced to enhance the security of NNs by preventing unauthorized access to the NN weights. The proposed architecture provides flexibility to configure the RRAM device in multiple modes to suit different applications. It shows promise in achieving a more efficient and compact design for the hardware implementation of NNs and security primitives.**

***Index Terms*—TRNG, PUF, NN, RRAM, Memristors, Hardware Security**

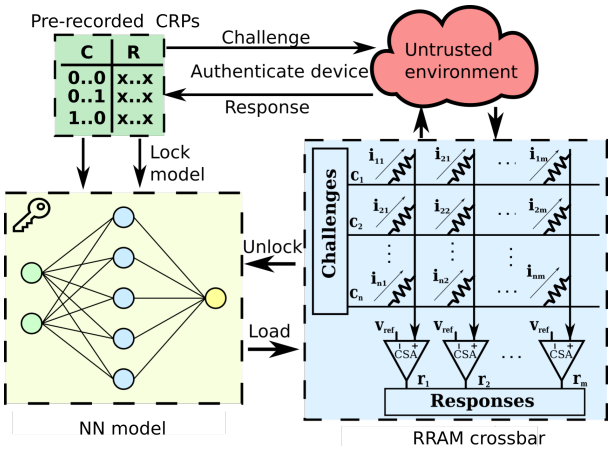


Fig. 1: The NN locking using the embedded hardware security primitives by exploiting the variations of RRAM cells.

I. Introduction

The Internet of things (IoT) era has led to a significant increase in data exchange between processors and memory, which results in high power consumption. It ultimately degrades the system's performance [1]. The von Neumann architecture, which separates processing and memory units, often suffers from data access latency due to the large volume of data movement [2]. To overcome these limitations, various novel computing paradigms are being investigated. In-memory computing, which performs calculations entirely within the computer memory, has gained significant attraction as a potential solution [3]–[5].

Additionally, it is necessary to authenticate IoT devices on the network to ensure data security and protection by maintaining their integrity, confidentiality, and availability, thus preventing any malicious attacks or unauthorized access. Physical unclonable function (PUF) circuits are becoming popular in IoT due to their ability to generate unique and unpredictable responses to challenges. This makes them highly useful for hardware security, such as device authentication and key generation, and for implementing security protocols ranging from device attestation to data encryption. Several circuits have been proposed for realizing in-memory computing architectures using resistive random access memory (RRAM) devices to implement various techniques, including PUF [6]–[8], neuromorphic neurons [9], [10] and digital gates [11]–[13]. RRAM is being considered as a potential candidate to address various drawbacks of conventional complementary metal oxide semiconductor (CMOS)-based architectures [14]. The relatively small size of RRAM devices also makes it highly feasible to integrate computing circuits and memory, thus realizing efficient architectures for learning algorithms, hardware security modules, and neural network (NN) applications [15].

RRAM has been extensively studied for main memory and in-memory computing architectures, but their stochastic nature and intrinsic variation in switching parameters have hindered their widespread adoption as next-generation memories [16]. However, this uncertain behavior is desirable for designing hardware security primitives [17]. Another commonly investigated hardware encryption module is true random number generators (TRNGs), which generate a stream of random numbers by exploiting randomness in physical processes [18]. While CMOS-based TRNG designs have been proposed, they only provide limited security-specific properties, paving the way for TRNGs based on emerging technologies. Among these designs, RRAM-based TRNGs demonstrate desirable properties, primarily due to their low power operation, high

§Equal contribution

density, and stochastic filament formation [19].

The protection of trained neural network (NN) models has become crucial to prevent unauthorized access, which can lead to the cloning of the model by adversaries. This study proposes a novel architecture to implement NN and hardware security modules on a single RRAM crossbar, allowing only authorized users with the correct device to use the locked NN model [20]. The proposed architecture focuses on protecting the intellectual property (IP) rights of deep NN models. Fig. 1 illustrates the framework for locking the NN using the embedded security module. The major contributions of this work are as follows:

- Integrating the NN, PUF, and TRNG on the RRAM crossbar array.
- Discussion on how the proposed crossbar architecture can be used for realizing NN weights locking.
- Lastly, the methodology for implementing the NN, PUF, and TRNG on the same crossbar is validated.

The remainder of the paper is organized as follows: Section II details the architecture for implementing NN, TRNG, and PUF designs based on RRAM. Section III shows the NN weights-locking algorithm using the proposed architecture. Section IV discusses the results of integrating NN and hardware security primitives realized using the crossbar RRAM array. Section V concludes the paper.

II. Proposed Architecture

This section explains the architecture to integrate NN and hardware security modules using the RRAM crossbar. The proposed architecture is shown in Fig. 2. The RRAM cells connected in a passive crossbar configuration are at the core of the proposed architecture. The same crossbar has been used to implement NN, PUF, and TRNG.

A. RRAM crossbar

The RRAM device employed in this investigation is characterized by its ability to store multiple bits. Specifically, the device can be programmed into a low resistive state (LRS) and a high resistive state (HRS). Still, it can also store multiple resistive states between LRS and HRS, which is referred to as multi-state storage. Applying varying voltage pulses across the device can be configured as two- or multi-state devices. For the purpose of the vector-matrix multiplication (VMM) implementation in this study, the device is configured as a multi-state device, but it can also function in the two-state mode. The proposed architecture offers the flexibility to configure the device in multiple modes to suit applications such as VMM, TRNG, and PUF implementations. Next, we will discuss using RRAM as a core device to implement these applications on a single crossbar.

B. VMM implementation

The RRAM crossbar performs the VMM operation, a critical function in implementing NNs. The weight matrix required for VMM is stored on a crossbar, with weights corresponding to each device's resistance or conductance state.

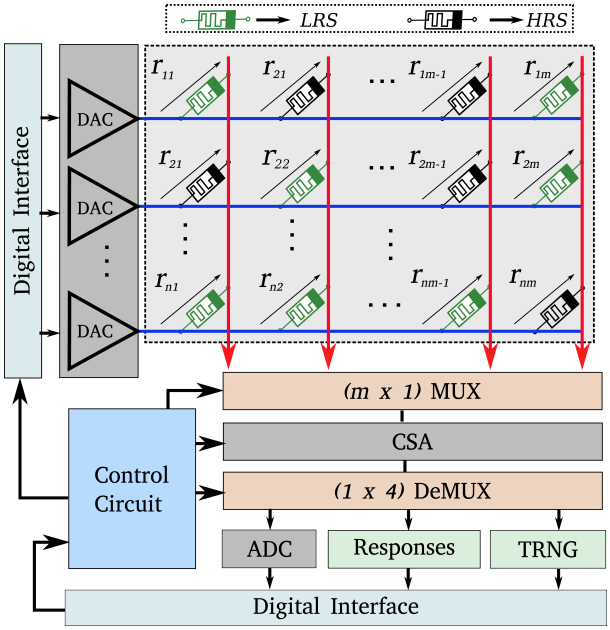


Fig. 2: Proposed architecture for implementing NNs and hardware security primitives on the crossbar of RRAM.

These devices are set up to store multi-bit weights, and their rows are linked to a digital-to-analog converter (DAC). The input vector is then fed into the DACs, which generally convert the input into analog output voltage levels, as depicted in Fig. 2. Following Ohm's current law, the applied input voltages produce a current through each device depending on the device's resistance or conductance value (weight). The current flowing through each device connected in a column is combined based on Kirchhoff's current law. Ultimately, the current values in the columns are utilized to carry out a multiplication and accumulation operation of the weights and input vector, which is nothing but a VMM operation in memory.

$$\begin{matrix} v_1 \\ v_2 \\ v_2 \\ v_0 \end{matrix}\begin{pmatrix}1\\2\\2\\0\end{pmatrix}^T \begin{matrix} \\ r_0 \\ r_1 \\ r_2 \\ r_3 \end{matrix}\begin{matrix} c_0 & c_1 & c_2 & c_3 \end{matrix}\begin{pmatrix}1 & 2 & 3 & 3\\0 & 3 & 0 & 1\\2 & 2 & 0 & 1\\3 & 2 & 2 & 1\end{pmatrix} = \begin{matrix} c_0 & c_1 & c_2 & c_3 \end{matrix}\begin{pmatrix}5 & 12 & 3 & 7\end{pmatrix} \quad (1)$$

To illustrate, consider the VMM operation involving two-bit weights and two-bit input vectors. The input and weights matrix has been shown in Equation 1. DAC maps the input vector to the voltage levels from v_0 to v_3. The weights matrix is stored as a resistance state on the crossbar (marked in Equation 1 using r_n to c_m). The resistance state of the device in the crossbar represents the two-bit weights, while the input vector is applied to the DAC through a digital interface, as shown in Fig. 2. The multiplication and accumulation results are obtained for each column, and the current sense amplifier for digital conversion amplifies the resulting current. In the example execution, the output matrix contains results greater

than two bits, which need to be converted back to the digital domain. The analog-to-digital (ADC) resolution is determined by $\lceil \log_2(w \times m) \rceil$, where w represents the weight bits (2 in this example) and m means the number of devices in a single column (4 in this example).

C. TRNG

To implement the TRNG on the RRAM crossbar, we have used the technique presented in [21], where the device-to-device (D2D) and cycle-to-cycle (C2C) variations on the crossbar have been used to generate the random switching in the crossbar. The switching threshold of each device on the RRAM crossbar is used to generate random bits. By applying a 50% switching probability pulse to the crossbar, random devices switch their state to LRS, and the others remain in the HRS. Due to the crossbar variation, each device's switching threshold is different, resulting in random switching of the devices. In order to implement the TRNG in the proposed architecture, another terminal of the device must connect to GND, which the 1x4 DeMUX controls.

D. PUF

A single RRAM crossbar is utilized in this proposed architecture to implement a TRNG and a PUF. The crossbar is first initialized to an entropy source based on the TRNGs algorithm, which provides a source of randomness to generate random bits. Challenges are then applied to the crossbar's rows, and the responses of the PUF are collected. The challenges are mapped to a read voltage pulse, which varies based on the input challenge. However, during PUF implementation, the crossbar is configured to two-state rather than multi-state switching.

To collect the responses of the PUF, Kirchoff's current law is applied to the crossbar, which collects the current flowing through each device at the column lines. The input challenge and device variations influence this current. The sneak path affects the current in the crossbar, which contributes to the current at each column in a completely random manner. The analog current values are converted to boolean response bits at the output using a current sense amplifier (CSA). As the response bit can be either 0 or 1, ADC in the path has been bi-passed using 1x4 DeMUX. The digital interface can further use the collected responses to lock the weights matrix on the crossbar. The randomness and unpredictability of the PUF response make it suitable for use in secure authentication and key generation applications, and the incorporation of TRNG adds a layer of security.

III. Weights locking

Weights locking is a required method used to safeguard the intellectual property of NN models, particularly in scenarios where the model has been trained on sensitive data or where the model's performance is essential to business success. The proposed architecture integrates a PUF as a hardware security module in the RRAM crossbar. During training, the weights are encrypted using a unique key generated by the PUF. The architecture is configured to implement the PUF and generate the key to encrypt the weights. The encrypted weights and the challenge are then provided to the user.

During the inference process, the architecture is reconfigured to implement the PUF, and the challenge provided with the encrypted weights is applied to generate the key. Next, the encrypted weights are loaded into the NN, and the key generated by the PUF is utilized to decrypt the weights. The decrypted weights are then used to make predictions.

A. Locking

The suggested design enables the NN to be secured within an RRAM crossbar. With the hardware security module situated on the same crossbar, a key can be generated via configuration in both the TRNG and PUF setups. The PUF-generated key can then be used to encrypt the weights to be stored in the crossbar. Algorithm 1 outlines the use of the proposed architecture for NN weights locking.

Algorithm 1 An algorithm for weights locking

1: Choose TRNG implementation
2: Apply 50% probability switching pulse
3: Choose PUF implementation
4: Apply challenge and collect the responses (key)
5: Store the CRPs on the server side and use a key to encrypt the weights.
6: Send the encrypted weights to sharing platform with the challenge

B. Unlocking

Once the weights have been encrypted, they can be transmitted to the user via any sharing platform. An identical challenge will be employed on the device's end to generate the required key. Algorithm 2 specifies the decryption procedure. The device's inherent randomness is expressed via CRPs, which are unique to each device, which helps prevent attackers from using the same weights on a different hardware device.

Algorithm 2 An algorithm for unlocking the weights

1: Receive the encrypted weights
2: Choose TRNG implementation
3: Apply 50% probability switching pulse
4: Choose PUF implementation
5: Apply challenge and generate the key
6: Choose the key to decrypt the weights
7: Store the decrypted weights on the same crossbar (overwrite the TRNG entropy)

In summary, this paper describes an architecture that integrates NNs and hardware security modules using passive RRAM cells connected in a crossbar structure. The RRAM device can store multiple resistive states between low and high resistive states, allowing it to function as a two-state or multi-state device. The proposed architecture offers flexibility to configure the device in multiple modes to suit different applications, such as VMM, TRNG, and PUF.

IV. Experimental Results

For this study, an RRAM cell comprises a Pt/Ti/TiO_x/HfO_2/Pt material stack, demonstrating improved stability in terms of both electroforming voltage and thermal [22]. This device can be configured into different configurations, such as binary and multi-state switching. The device is programmed into resistance by applying a voltage pulse with a specific duration and amplitude. The device exhibits resistance between $60-100K\Omega$ in HRS and $1.5-1.6K\Omega$ in LRS. However, in a multi-state configuration, there can be multiple states between HRS and LRS. The devices in the crossbar are utilized without any selector (passive) in series. Passive crossbars typically face the issue of sneak-path current, which can be utilized to design TRNGs and PUFs.

A. Switching and Variations

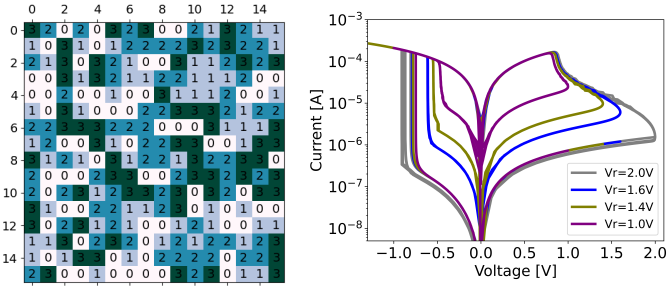
In order to switch the device between binary states, a 150ns pulse of 2.0V with 10ns rise and the fall time is applied to switch the device to LRS (programming to 1), while a negative pulse of 2.0V is applied to switch the device to HRS (programming to 0). However, a gradual RESET method has been employed to achieve multi-state behavior. In this method, the device is initialized to LRS, and then an incremental pulse is applied to switch it to multiple states. The I-V curves of multi-state switching have been shown in Fig. 3.

The switching of the devices can be affected by variations in D2D and C2C parameters. These variations are influenced by manufacturing variations in device radius, device length, and oxygen ion concentration in the dielectric. As a result of these variations, the HRS resistance varies from $31K\Omega$ to $155K\Omega$ with an average of $65.56K\Omega$, while the LRS resistance varies from $1.55K\Omega$ to $1.67K\Omega$ with an average of $1.64K\Omega$. However, the HRS has a wide distribution; all HRS values are distinguishable from LRS. By carefully selecting the pulse to RESET the devices, they can be switched randomly from LRS to HRS or vice versa, which can be exploited to design the TRNGs.

B. PUF properties

We conducted extensive experiments to evaluate the reliability and performance of PUF on the proposed architecture. Our results demonstrate a reliability of 100%, indicating that the CRPs generated by the proposed PUF are consistent and repeatable across multiple trials. Additionally, the uniqueness of the CRPs was found to be 47.78%, meaning that the probability of generating the same CRP for two different devices is low. The uniformity of the CRPs was measured to be 49.79%, indicating that the distribution of the CRPs is approximately uniform. Finally, the bit-aliasing was found to be 48.57%, indicating that the probability of generating the same CRP for two different challenges is also low.

Importantly, our results show that the proposed PUF achieves these performance metrics without needing post-processing techniques, making it a practical and efficient hardware security primitive. Overall, our findings demonstrate the potential for using RRAM-based PUF designs in hardware security applications with high reliability and security characteristics.



(a) (b)

Fig. 3: The mapping of 2-bit weights to the crossbar is demonstrated in (a), while the multi-state behavior of devices for analog weight storage is illustrated in (b).

C. NN

The crossbar can be utilized to map NN weights, similar to VMM applications. Fig.3a shows the mapping of 2-bit weights to the crossbar. To enable the multi-state behavior of a device, a gradual RESET method is employed, as illustrated in Fig.3b. We conducted a proof-of-concept by implementing VMM multiplication on a 16x16 crossbar, which is a critical operation for any NN implementation. The device's variations result in error accumulation at the input of the ADC. Nonetheless, this issue can be resolved through onboard fault-aware training of the required application.

V. Conclusions

In conclusion, this paper described a proposed architecture integrating NNs and hardware security modules using the RRAM crossbar as the core device. The proposed architecture enables a single crossbar to implement NN, TRNG, and PUF applications. The RRAM crossbar's multi-state storage characteristic is exploited to perform the vector-matrix multiplication operations required for NN implementation. The TRNG uses the crossbar's variation in device switching thresholds to generate random bits. The PUF is implemented using the same crossbar initialized as an entropy source for the TRNG. The proposed architecture provides flexibility to configure the RRAM device in multiple modes to suit different applications. This paper also presents the algorithms for NN weight locking. Overall, the proposed architecture shows promise in achieving a more efficient and compact design for the hardware implementation of NNs and security primitives.

Acknowledgments

This work was supported in part by the Federal Ministry of Education and Research (BMBF, Germany) in the project NEUROTEC II under Project 16ME0398K, Project 16ME0399 and through Dr. Suhas Pai Donation Fund at IIT Bombay.

REFERENCES

[1] W. Chen, Z. Qi, Z. Akhtar, and K. Siddique, "Resistive-ram-based in-memory computing for neural network: A review," *Electronics*, vol. 11, no. 22, p. 3667, 2022.

[2] Y. Long, T. Na, and S. Mukhopadhyay, "Reram-based processing-in-memory architecture for recurrent neural network acceleration," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 26, no. 12, pp. 2781–2794, 2018.

[3] D. Ielmini and H.-S. P. Wong, "In-memory computing with resistive switching devices," *Nature electronics*, vol. 1, no. 6, pp. 333–343, 2018.

[4] M. A. Zidan, J. P. Strachan, and W. D. Lu, "The future of electronics based on memristive systems," *Nature electronics*, vol. 1, no. 1, pp. 22–29, 2018.

[5] F. Staudigl, F. Merchant, and R. Leupers, "A survey of neuromorphic computing-in-memory: Architectures, simulators, and security," *IEEE Design & Test*, vol. 39, no. 2, pp. 90–99, 2022.

[6] H. Nili, G. C. Adam, B. Hoskins, M. Prezioso, J. Kim, M. R. Mahmoodi, F. M. Bayat, O. Kavehei, and D. B. Strukov, "Hardware-intrinsic security primitives enabled by analogue state and nonlinear conductance variations in integrated memristors," *Nature Electronics*, vol. 1, no. 3, pp. 197–202, 2018.

[7] R. A. John, N. Shah, S. K. Vishwanath, S. E. Ng, B. Febriansyah, M. Jagadeeswararao, C.-H. Chang, A. Basu, and N. Mathews, "Halide perovskite memristors as flexible and reconfigurable physical unclonable functions," *Nature Communications*, vol. 12, no. 1, p. 3681, 2021.

[8] S. Singh, S. Bodapati, S. Patkar, R. Leupers, A. Chattopadhyay, and F. Merchant, "Pa-puf: A novel priority arbiter puf," in *2022 IFIP/IEEE 30th International Conference on Very Large Scale Integration (VLSI-SoC)*, 2022, pp. 1–6.

[9] W. Huang, X. Xia, C. Zhu, P. Steichen, W. Quan, W. Mao, J. Yang, L. Chu, and X. Li, "Memristive artificial synapses for neuromorphic computing," *Nano-Micro Letters*, vol. 13, pp. 1–28, 2021.

[10] I. Boybat, M. Le Gallo, S. Nandakumar, T. Moraitis, T. Parnell, T. Tuma, B. Rajendran, Y. Leblebici, A. Sebastian, and E. Eleftheriou, "Neuromorphic computing with multi-memristive synapses," *Nature communications*, vol. 9, no. 1, p. 2514, 2018.

[11] J. Reuben, R. Ben-Hur, N. Wald, N. Talati, A. H. Ali, P.-E. Gaillardon, and S. Kvatinsky, "Memristive logic: A framework for evaluation and comparison," in *2017 27th International Symposium on Power and Timing Modeling, Optimization and Simulation (PATMOS)*. IEEE, 2017, pp. 1–8.

[12] D. S. Jeong, K. M. Kim, S. Kim, B. J. Choi, and C. S. Hwang, "Memristors for energy-efficient new computing paradigms," *Advanced Electronic Materials*, vol. 2, no. 9, p. 1600090, 2016.

[13] S. Balatti, S. Ambrogio, and D. Ielmini, "Normally-off logic based on resistive switches—part i: Logic gates," *IEEE transactions on Electron Devices*, vol. 62, no. 6, pp. 1831–1838, 2015.

[14] S. Mittal, "A survey of architectural techniques for improving cache power efficiency," *Sustainable Computing: Informatics and Systems*, vol. 4, no. 1, pp. 33–43, 2014.

[15] D. Soudry, D. Di Castro, A. Gal, A. Kolodny, and S. Kvatinsky, "Memristor-based multilayer neural networks with online gradient descent training," *IEEE transactions on neural networks and learning systems*, vol. 26, no. 10, pp. 2408–2421, 2015.

[16] F. Zahoor, T. Z. Azni Zulkifli, and F. A. Khanday, "Resistive random access memory (rram): an overview of materials, switching mechanism, performance, multilevel cell (mlc) storage, modeling, and applications," *Nanoscale research letters*, vol. 15, no. 1, pp. 1–26, 2020.

[17] Y. Pang, H. Wu, B. Gao, N. Deng, D. Wu, R. Liu, S. Yu, A. Chen, and H. Qian, "Optimization of rram-based physical unclonable function with a novel differential read-out method," *IEEE Electron Device Letters*, vol. 38, no. 2, pp. 168–171, 2017.

[18] R. Govindaraj, S. Ghosh, and S. Katkoori, "Csro-based reconfigurable true random number generator using rram," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 26, no. 12, pp. 2661–2670, 2018.

[19] B. Yang, D. Arumí, S. Manich, Á. Gómez-Pau, R. Rodríguez-Montañés, M. B. González, F. Campabadal, and L. Fang, "Rram random number generator based on train of pulses," *Electronics*, vol. 10, no. 15, p. 1831, 2021.

[20] A. Chakraborty, A. Mondai, and A. Srivastava, "Hardware-assisted intellectual property protection of deep learning models," in *2020 57th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2020, pp. 1–6.

[21] S. Singh, F. Zahoor, G. Rajendran, S. Patkar, A. Chattopadhyay, and F. Merchant, "Hardware security primitives using passive rram crossbar array: Novel trng and puf designs," in *Proceedings of the 28th Asia and South Pacific Design Automation Conference*, ser. ASPDAC '23. New York, NY, USA: Association for Computing Machinery, 2023, p. 449–454. [Online]. Available: https://doi.org/10.1145/3566097.3568348

[22] C. Bengel, A. Siemon, F. Cüppers, S. Hoffmann-Eifert, A. Hardtdegen, M. von Witzleben, L. Hellmich, R. Waser, and S. Menzel, "Variability-aware modeling of filamentary oxide-based bipolar resistive switching cells using SPICE level compact models," *IEEE TSCAS I*, vol. 67, no. 12, pp. 4618–4630, 2020.