Memristor Crossbar Design Framework for Quantum Computing

Abstract—**Over the last years there has been significant progress in the development of quantum computers. It has been demonstrated that they can accelerate the solution of various problems exponentially compared to today's classical computers, harnessing the properties of superposition and entanglement, two resources that have no classical analog. Since quantum computer platforms that are currently available comprise only a few tenths of qubits, as well as the access to a fabricated quantum computer is time limited for the majority of researchers, the use of quantum simulators is essential in developing and testing new quantum algorithms. Taking inspiration of our previous work on developing a novel quantum simulator based on memristor crossbar circuits, in this work we propose a Framework that automates the circuit design of emulated quantum algorithms. The proposed design framework covers the procedure of generating and initializing the memristor crossbar configuration that comprises the desirable quantum circuit, leading to a technology agnostic design tool. To such degree, different quantum algorithms can be efficiently emulated on memristor crossbar configurations for various types of memristive device, aiming to assist and accelerate the fabrication process of a memristor based quantum simulator.**

I. Introduction

Quantum computers use properties of quantum mechanics to perform computations that are intractable, or practically impossible for classical computers and even high-performance computation systems [1]. A wide variety of scientific problems, such as simulating quantum mechanical processes in physics, chemistry and biology, performing database search [2] and prime number factorization [3] can be efficiently solved. Quantum computers' superiority in parallel processing of data comes down to their unprecedented quantum properties, i.e. state superposition and quantum entanglement [4], two quantum mechanical resources with no classical analog.

To be able to investigate further various problems that quantum computers could provide us with an efficient solution, an important step is the development of quantum simulators [5], which not only contributed to the development of a wide variaty of quantum algorithms but are also an indispensable part of the quantum computer stack, where they serve as a classical/quantum interface between programming and execution of algorithms on quantum computing platforms [6]. Quantum simulators are software programs that run on classical computers and act as the target machine for a simulated quantum algorithm, making it possible to run and test quantum programs in an environment that predicts how qubits will react to different operations.

In quantum simulators qubits are represented by vectors in Hilbert space and quantum gates are Hilbert space operators represented by matrices. The effect of the Quantum gates on qubits and quantum registers is described by matrix-vector multiplications.

Quantum simualators such as IBM's $Qiskit$ [7] and Microsoft's $LIQUi| >$ [8] simulators are designed to execute the quantum algorithms on conventional computers. Our approach is aiming to allocate the simulation task to an unconventional in-memory computing capable hardware. Utilizing the designed quantum simulator framework, quantum algorithms can be executed on memristor crossbar configuration in an analog manner, which further reduces the complexity and scale of the hardware, while highly increasing speed through parallelization of the computations.

Memristor [9] are able to perform effectively matrix-vector multiplication in a crossbar configuration. Memristors are characterized by their non-volatility, high density and low power consumption. They are two terminal devices which resistance is controlled by an applied voltage signal across its terminals, while also depended on its state's history. We have studied the use of memristor circuits as part of the quantum computing stack, where they also act as accelerators, and here we propose a framework that is capable of generating different memristor crossbar configurations that, after proper intitialization of their memristor conductances, can effectively implement quantum computations.

II. Quantum Computations utilizing Memristor Crossbars

Generally, the steps to implement a quantum computations are the initialization of the qubits in base states, performing various unitary operations and finally measuring the qubit states. The main operation during the quantum computation is the matrix-vector multiplication which corresponds to Quantum Gate (unitary Operator)-Qubits interaction.

Concerning the representation of a qubit state, it can be performed by a column vector:

$$= \alpha\,|0\rangle + \beta\,|1\rangle = \alpha \begin{bmatrix} 1 \\ 0 \end{bmatrix} + \beta \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} \alpha \\ \beta \end{bmatrix} \tag{1}$$

where $|0\rangle$ and $|1\rangle$ are the basis vectors and α and β the corresponding probability amplitudes, where $|\alpha|^2 + |\beta|^2 = 1$ has to be fulfilled. In the same manner, the action of quantum gates on qubits can be represented by the corresponding matrix and performed through a matrix-vector multiplication. For example, the action of a Hadamard gate on a qubit, which maps $|0\rangle \rightarrow \frac{|0\rangle+|1\rangle}{\sqrt{2}}$ and $|1\rangle \rightarrow \frac{|0\rangle-|1\rangle}{\sqrt{2}}$, can be performed as:

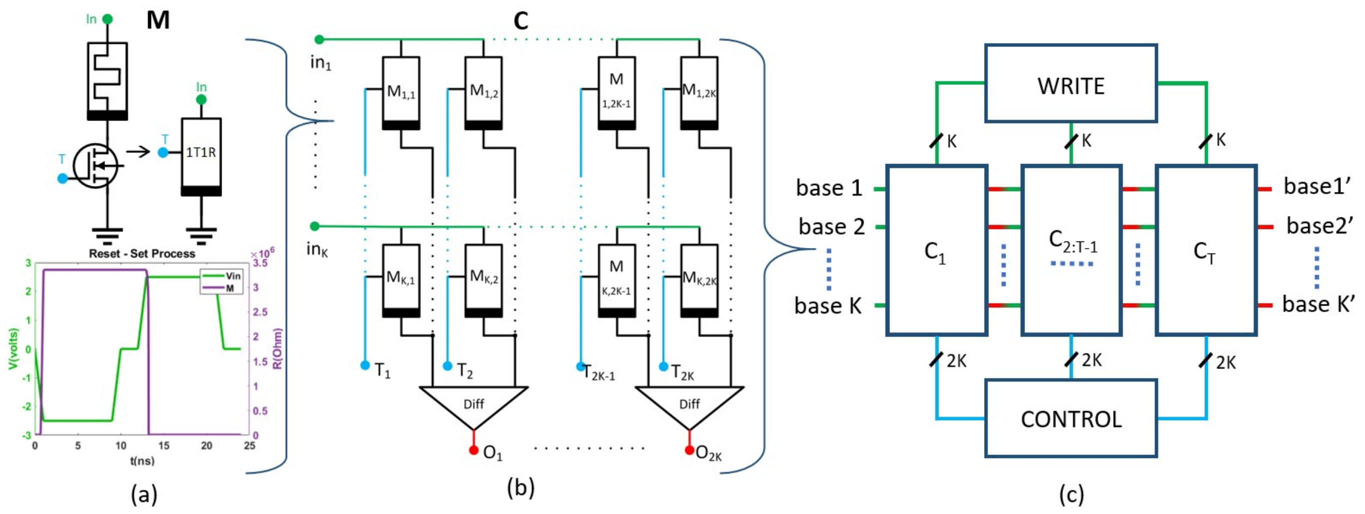


Fig. 1. Quantum Computation capable Crossbar Circuit (a) utilized 1T1R (b) Differential Crossbar (c) Memristor based Quantum Circuit

$$H\left|0\right\rangle = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}\begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{bmatrix} \tag{2}$$

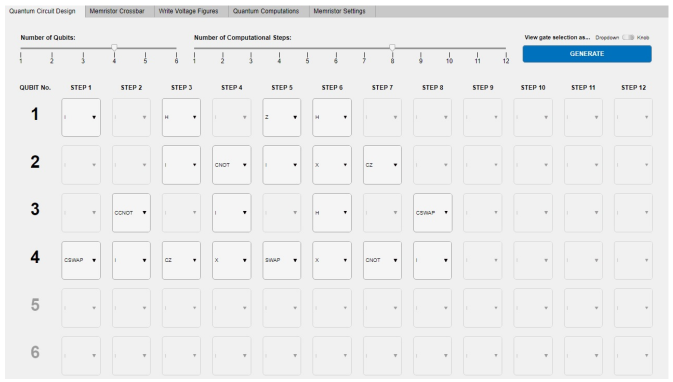
$$H\left|1\right\rangle = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}\begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} \frac{1}{\sqrt{2}} \\ -\frac{1}{\sqrt{2}} \end{bmatrix} \tag{3}$$

A memristor crossbar configuration is capable of effectively performing matrix-vector multiplication. In particular, by exploiting the cross-point structure, a memristor crossbar array uses the applied row voltages as the input vector X, the memristors' conductance as values of the weight matrix W and the column current as the output vector. Consequently, the array realizes the matrix-vector multiplication efficiently with $O(1)$ time complexity, as the cross-point currents are accumulated instantly in each column, providing the desired output naturally through Ohm's law [10]–[12]. Taking advantage of the reprogrammability of memristors, different quantum gates can be implemented utilizing the same array.

In this framework, taking inspiration by our previous work, a generalized approach of N Qubits and T steps has been adopted. In further detail, the implementation of the quantum computation for the N-qubit case, requires a set of $K \times K$ unitary matrices, where $K = N^2$ is the number of the base states that can be represented by N number of Qubits. In these matrices only ternary values are utilised ($\{-1, 0, 1\}$), along with the multiplying coefficients, required for some matrices, i.e. in the Hadamard gate. Concerning the circuit implementation of the crossbar array design, the ternary values are represented by two binary memristor devices in different columns, so a $N^2 \times 2N^2$ differential memristive crossbar is used for quantum gate (step), as shown in Fig.1(b). Moreover, the necessary coefficients, which are multiplied with the whole matrix, can be encoded in the gain of the output differential amplifier of the crossbar, mitigating the problem of the exact programming of memristors to the design of conventional circuit elements.

III. Graphical User Interface (GUI) of the Framework

A Graphical User Interface (GUI) has been designed, in order to provide the user with the ability to implement and examine universal quantum computations on memristor crossbars. The initial screen of the GUI (Fig. 2) illustrates all the available options for the quantum computations. The available options of the simulator include the number of needed qubits, the number of computational steps, and the gates applied at each computational step. The leftmost column represents the row number of each qubit and every other column represents the quantum gates applied at each computational step. The user can enter a variety of quantum gates such as, Hadamard, Toffoli, and Fredkin gates, as well as numerous Pauli and controlled gates. The user can generate the needed circuit by clicking the "GENERATE" button after making the corresponding selections.



Number of Qubits: 1 2 3 4 5 6

Number of Computational Steps: 1 2 3 4 5 6 7 8 9 10 11 12

View gate selection as... Dropdown Knob

QUBIT No.	STEP 1	STEP 2	STEP 3	STEP 4	STEP 5	STEP 6	STEP 7	STEP 8	STEP 9	STEP 10	STEP 11	STEP 12
1	I	I	H	I	Z	H	I	I	I	I	I	I
2	I	I	I	CNOT	I	X	CZ	I	I	I	I	I
3	I	CCNOT	I	I	I	H	I	CSWAP	I	I	I	I
4	CSWAP	I	CZ	X	SWAP	X	CNOT	I	I	I	I	I
5	I	I	I	I	I	I	I	I	I	I	I	I
6	I	I	I	I	I	I	I	I	I	I	I	I

Fig. 2. Initial screen of the Graphical User Interface (GUI)

When the needed circuit is generated, the user can view more details by browsing in the uppermost tabs, namely ”Memristor Crossbar”, ”Write Voltage Figures”, and ”Quantum Computations”. In the first of the mentioned tabs, the simulated memristor crossbar is illustrated along with all the memristances after the crossbar programming is performed. In the tab after, the input voltage (Write Plot) and the transistor gate voltage (Selector Plot) are illustrated for the WRITE process for each computational step separately. In the ”Quantum Computations” tab, the user can select the initial state of the qubit in each row and perform all the available computations, which are illustrated as a figure of the output coefficients in each qubit base state, by clicking the ”RUN” button.

The GUI has been developed in such way to be easily adapted not only in different selections of the quantum circuit, but also in different memristor devices. Thus, a last tab is available, namely ”Memristor Settings”, for the user to select all the memristor model parameters that fit the available device, as well as the pulse parameters and the simulation time step needed for successful simulations. As it comes to mind, the simplicity and the versatility of the developed GUI along with the adaptability and the reprogrammibility of the memristor crossbar can be easily used for a wide-range of quantum applications.

The aforementioned GUI has been designed in MATLAB app designer, nevertheless a standalone installation is provided in the supplementary files, consequently the user can optionally execute the framework without the need of an installed MATLAB software.

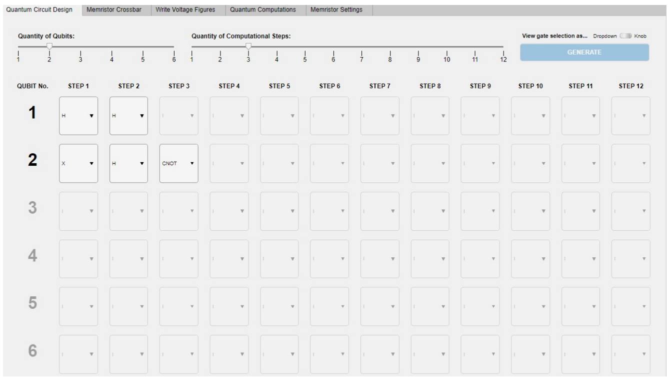


Fig. 3. Quantum Circuit selection

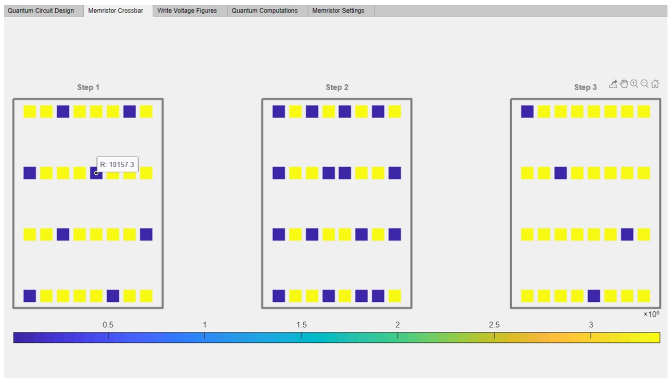


Fig. 4. Generated Crossbar Configuration of each step

IV. From GUI to Crossbar Generation, Initialization and Quantum Computation

The GUI and the underlying circuit design algorithm are in constant communication. Initially, the GUI provides the necessary information to the algorithm such as the number of the Qubits and the sequence of the applied Quantum Gates represented in matrix format. Utilizing this information the ciruit design algorithm generates the multiple memristor crossbar configuration. Based on the number of Qubits N and the number of steps T, T Crossbars of $N^2 \times 2N^2$ size are generated and connected in series.

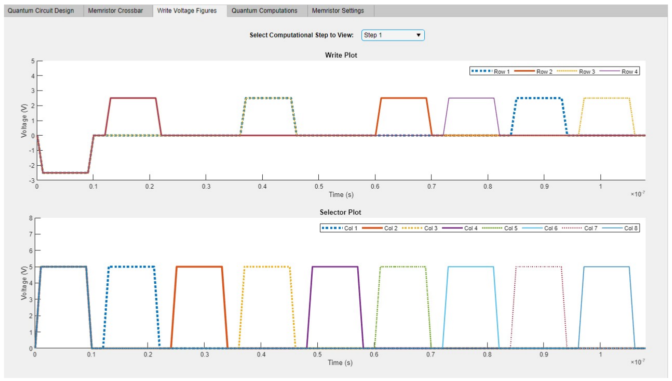
Consequently, each crossbar is initialized through a write signal programming, to represent the corresponding step. Initially all the columns are activated by setting to high both the Selector Gate Voltage T_1-T_{2K}, as well as, the input signals (in_1-in_K) of Fig.1b, in order to reset the whole crossbar to high resistance state. Next, through T_1-T_{2K} the appropriate memristors are initialized to the low resistance state column by column, by applying the in_1-in_K signals to the corresponding row. By keeping the number of Qubits and the number of the steps constant, different quantum circuits can be implemented on the same crossbar configurations just by re-initializing the memristor crossbars.

During the quantum computation the initial state of the Qubits is set by the user. The GUI propagates the user’s selection to the algorithm which in turn scales the coefficients of the Qubits base states to the range of $0 - 100mV$. This way the computation will be performed without the voltage affecting the conductance values of the memristors. The output voltage vector of the final memristor crossbar corresponds to the final Qubit state which is the result of the quantum computation.

The simulated crossbar consists of 1T1R (transistor in series with memristor) devices shown in Fig. 1a. The memristor model utilized in the framework is the ASU RRAM Compact Model [13]. It is a physics-based RRAM device model with compact equation set, developed in Verilog-A that enables the large scale circuit simulations. The model properly captures the device’s change in resistance due to the filamentary switching found in fabricated RRAM devices and it is fitted to the experimental data of IMEC HfO_x-based RRAM devices [14], [15] when left in default settings, which are able for fast transitions between their resistance states ($10K\Omega$-$3M\Omega$) in the range of nanoseconds. The framework provides the user with the ability to change the parameters of the memristor model to be able to reflect the response of other fabricated devices, as well as, the amplitude and width of the write pulse that will be utilized in the programming of the memristor crossbars.

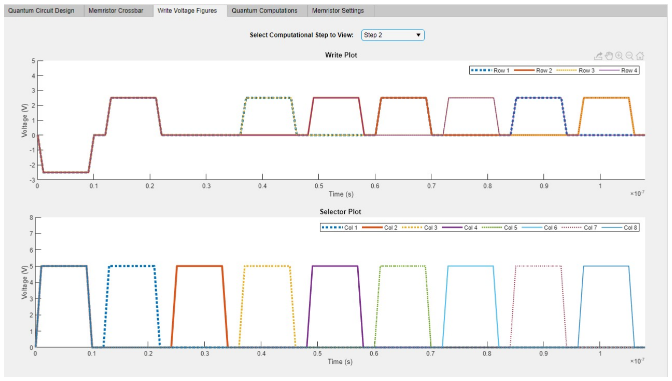
V. Simulation Results of the Framework

In order to demonstrate the functionality and features of the framework, a quantum computation example is performed. In



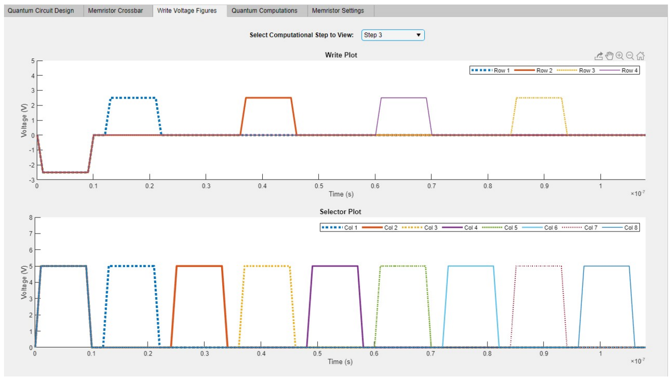
Select Computational Step to View: Step 1

(a)



Select Computational Step to View: Step 2

(b)



Select Computational Step to View: Step 3

(c)

Fig. 5. Memristor crossbar initialization programming of (a) Step 1: $H \otimes X$ (b) Step 2: $H \otimes H$ (c) Step 3: $CNOT$

this example a two Qubit quantum circuit of three steps is implemented. The equation corresponding to the aforementioned circuit can be examined below:

$$|\psi\rangle = CNOT(H \otimes H)(H \otimes X)\,|Q\rangle \tag{4}$$

In the Quantum Circuit Design tab of Fig.3 the gates that correspond to the (4) are selected. By executing the circuit generation algorithm, the crossbars that constitute the quantum algorithm are generated and visualised in the Memristor Crossbar tab of Fig.4 through which, the resistance value of each memristor can be examined.

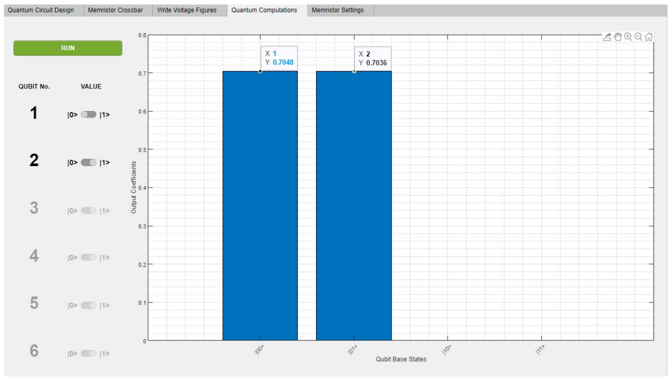


Fig. 6. Quantum Computation Results for the case of input $|01\rangle$

In the "Write Voltage Figures" tab of Fig.5, the write procedure of each crossbar can be observed by examining the applied row and selector signals for each step. $Step$ 1 corresponds to the $(H \otimes X)$ quantum operator, while $(H \otimes H)$ and $CNOT$ correspond to $Step$ 2 and 3 accordingly. For the write operation of this example, 10ns pulses of 2.5V amplitude with a rise and fall time of 1ns have been applied.

Finally the results of the quantum computation can be observed in the following tab of Fig.6. The output voltage vector of $Step3$ is shown in this figure, that corresponds to the result of the implemented memristor based quantum circuit, for the case of input $|01\rangle$. The output vector of the memristor crossbar circuit correspond to the output coefficients of the base states shown in (5).

$$|\psi_{01}\rangle = 0.7048\,|00\rangle + 0.7036\,|00\rangle + 0.0004\,|00\rangle + 0.0004\,|00\rangle \tag{5}$$

Comparing the circuit's result to the mathematical quantum calculation, it can be examined that utilizing ASU RRAM model fitted to IMEC HfO$_x$ devices and programming it with the before-mentioned pulses, an error rate of 0.4% can be achieved. On the same manner, every other input combination of qubits' base state can be tested.

By examining each tab, the user can extract valuable information on how to more accurately initialize the memristor crossbars and which memristors are better suited to perform the quantum computation. That information can contribute on further improving the memristor crossbar implementation.

VI. Conclusions

In this work a framework has been implemented capable of automatically generating memristor crossbar configurations that can perform quantum computations. This tool aims to simplify the investigation and the fabrication of a memristor based hardware capable of efficiently simulating a quantum computer. As a future version of the framework the variability of the memristors will be taken under consideration, as well as, a variety of memristor models will be integrated.

References

[1] T. Ladd, F. Jelezko, R. Laflamme, Y. Nakamura, C. Monroe, and J. O'Brien, "Quantum computers," *Nature*, vol. 464, pp. 45–53, 2010.

[2] L. K. Grover, "A fast quantum mechanical algorithm for database search," in *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, 1996, pp. 212–219.

[3] P. W. Shor, "Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer," *SIAM review*, vol. 41, no. 2, pp. 303–332, 1999.

[4] M. A. Nielsen and I. Chuang, *Quantum computation and quantum information*. Cambridge University Press, 2010.

[5] T. H. Johnson, S. R. Clark, and D. Jaksch, "What is a quantum simulator?" *EPJ Quantum Technology*, vol. 1, no. 1, p. 10, Jul 2014.

[6] I. Buluta and F. Nori, "Quantum simulators," *Science*, vol. 326, pp. 108–111, 2009.

[7] G. Aleksandrowicz, T. Alexander, P. Barkoutsos, L. Bello, Y. Ben-Haim, D. Bucher, F. Cabrera-Hernández, J. Carballo-Franquis, A. Chen, C. Chen *et al.*, "Qiskit: An open-source framework for quantum computing," *Accessed on: Mar*, vol. 16, 2019.

[8] D. Wecker and K. M. Svore, "Liqui—¿: A software design architecture and domain-specific language for quantum computing," *arXiv preprint arXiv:1402.4467*, 2014.

[9] D. B. Strukov, G. S. Snider, D. R. Stewart, and R. S. Williams, "The missing memristor found," *Nature*, vol. 453, no. 7191, p. 80, 2008.

[10] M. Hu, J. P. Strachan, Z. Li, E. M. Grafals, N. Davila, C. Graves, S. Lam, N. Ge, J. J. Yang, and R. S. Williams, "Dot-product engine for neuromorphic computing: Programming 1t1m crossbar to accelerate matrix-vector multiplication," in *Proceedings of the 53rd annual design automation conference*. ACM, 2016, p. 19.

[11] L. Xia, P. Gu, B. Li, T. Tang, X. Yin, W. Huangfu, S. Yu, Y. Cao, Y. Wang, and H. Yang, "Technological exploration of rram crossbar array for matrix-vector multiplication," *Journal of Computer Science and Technology*, vol. 31, no. 1, pp. 3–19, 2016.

[12] C. Li, M. Hu, Y. Li, H. Jiang, N. Ge, E. Montgomery, J. Zhang, W. Song, N. Dávila, C. E. Graves *et al.*, "Analogue signal and image processing with large memristor crossbars," *Nature Electronics*, vol. 1, no. 1, p. 52, 2018.

[13] X. Guan, S. Yu, and H.-S. P. Wong, "A spice compact model of metal oxide resistive switching memory with variations," *IEEE electron device letters*, vol. 33, no. 10, pp. 1405–1407, 2012.

[14] Y. Y. Chen, B. Govoreanu, L. Goux, R. Degraeve, A. Fantini, G. S. Kar, D. J. Wouters, G. Groeseneken, J. A. Kittl, M. Jurczak *et al.*, "Balancing set/reset pulse for $> 10^{10}$ endurance in hfo 2/hf 1t1r bipolar rram," *IEEE Transactions on Electron devices*, vol. 59, no. 12, pp. 3243–3249, 2012.

[15] Y. Y. Chen, M. Komura, R. Degraeve, B. Govoreanu, L. Goux, A. Fantini, N. Raghavan, S. Clima, L. Zhang, A. Belmonte *et al.*, "Improvement of data retention in hfo 2/hf 1t1r rram cell under low operating current," in *2013 IEEE International Electron Devices Meeting*. IEEE, 2013, pp. 10–1.