MemSPICE: Automated Simulation and Energy Estimation Framework for MAGIC-Based Logic-in-Memory

Simranjeet Singh*¶, Chandan Kumar Jha‡, Ankit Bende¶. Vikas Rana¶,
Sachin Patkar*, Rolf Drechsler‡§, Farhad Merchant†
*Indian Institute of Technology Bombay, India, ‡University of Bremen, Germany,
¶Forschungszentrum Jülich GmbH, Germany, §DFKI GmbH, Germany, †Newcastle University, UK
{simranjeet, patkar}@ee.iitb.ac.in, {si.singh, a.bende, v.rana}@fz-juelich.de,
{chajha, drechsler}@uni-bremen.de, farhad.merchant@newcastle.ac.uk

***Abstract*—Existing logic-in-memory (LiM) research is limited to generating mappings and micro-operations. In this paper, we present *MemSPICE*, a novel framework that addresses this gap by automatically generating both the netlist and testbench needed to evaluate the LiM on a memristive crossbar. MemSPICE goes beyond conventional approaches by providing energy estimation scripts to calculate the precise energy consumption of the testbench at the SPICE level. We propose an automated framework that utilizes the mapping obtained from the SIMPLER tool to perform accurate energy estimation through SPICE simulations. To the best of our knowledge, no existing framework is capable of generating a SPICE netlist from a hardware description language. By offering a comprehensive solution for SPICE-based netlist generation, testbench creation, and accurate energy estimation, MemSPICE empowers researchers and engineers working on memristor-based LiM to enhance their understanding and optimization of energy usage in these systems. Finally, we tested the circuits from the ISCAS'85 benchmark on MemSPICE and conducted a detailed energy analysis.**

***Index Terms*—Memristors, Digital Logic-in-Memory, MAGIC, Energy-Efficiency, SPICE Simulation**

I. Introduction

In-memory computing using memristors is gaining popularity as it helps overcome the von Neumann bottleneck in traditional computing. Memristors possess two states: high resistive state (HRS) and low resistive state (LRS), which are mapped to Boolean logic '0' and '1' for logic-in-memory (LiM) implementation. Among various techniques like IMPLY [1], FELIX [2], and majority logic [3], memristor-aided logic (MAGIC) [4] is widely adopted for LiM due to its superior energy and latency performance [5].

The larger design is synthesized to MAGIC NOR and NOT gates for a single-row memristor crossbar using the SIMPLER tool [6]. The tool maps MAGIC operations to three memristors (two inputs, one output) and two memristors (one input, one output) on the crossbar. Additionally, it generates the necessary number of cycles and input/output memristors for a given application or benchmark. As an illustration, consider a netlist for a half adder provided in Fig. 1 ❶. Initially, it is synthesized into a NOR and NOT netlist, as depicted in Fig. 1 ❷. Subsequently, the NOR and NOT gates are sequentially mapped onto memristors connected in series. The resulting implementation of MAGIC NOR and NOT using memristors is presented in Fig. 1 ❹.

Given the increasing popularity of the MAGIC design style in mainstream computing [5], evaluating this technique's energy consumption on larger circuit datasets is crucial. However, current methods for energy calculation rely on a coarse-grained approach, multiplying the average energy consumption of an operation by the number of occurrences in an application [7]. This approach is unsuitable for accurately estimating energy consumption in the MAGIC design style. A more detailed analysis of the implemented circuit is necessary to provide fine-grained energy values as presented in [8].

This paper proposes *MemSPICE*, an automated SPICE-level simulation and accurate energy estimation framework for the MAGIC design style. The proposed framework automatically generates a SPICE-level netlist and testbench voltages for a given application/benchmark. Furthermore, it provides fine-grained energy numbers by calculating the energy consumed by each device in the crossbar, irrespective of its contribution to the operation. The framework empowers researchers to obtain accurate energy estimates for digital designs, offering valuable insights into their methodologies at the circuit level. Furthermore, circuit designers can utilize this framework to implement additional optimizations at the circuit level, showcasing their benefits on the entire crossbar rather than just individual gates. The followings are the contributions of this paper:

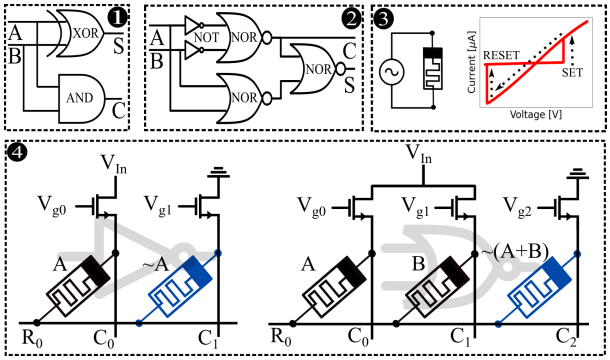


Fig. 1. Mapping process of standard logic gates defined in hardware description language to MAGIC NOT and NOR gates using memristors. The figure showcases the I-V curve of the memristors, providing insights into their resistive states and electrical characteristics.

- Introducing MemSPICE, a framework that takes SIMPLER tool mapping (*.json*) as input and autonomously generates the SPICE-level netlist.
- Detailed energy estimation techniques for MAGIC-based digital LiM at the low level, providing valuable insights to designers.
- Finally, the SPICE-level netlist for widely-used benchmarks (ISCAS'85) is automatically generated using MemSPICE, and a comparative analysis of the energy consumption values with current methodologies is conducted0.

The rest of the paper is organized as follows. Section II discusses the necessary background and related work. In Section III, we discuss the MemSPICE methodology in detail. In Section IV, we discuss the benchmark circuit preparation, the results obtained using MemSPICE methodology, and compare the results with state-of-the-art methods. We conclude the paper in Section V.

II. Background and Related Work

A. Memristive Devices or Memristors

Memristive devices or memristors are two-terminal passive devices with variable resistance. When a voltage is applied across the terminals of a memristor, the resistance changes in response to the

magnitude and direction of the current flow. Fig. 1 ❸ depicts the electrical characterization of a memristor, showcasing its I-V curve with distinct SET and RESET points marked. Importantly, even when the power is turned off, the memristor retains its resistance value until a new voltage is applied, making it a non-volatile memory element [9]. Due to their unique characteristics, memristors have garnered significant interest in various fields, such as in-memory computing, neuromorphic computing, and LiM applications. The maximum and minimum resistance values of memristors are represented by LRS and HRS, which are mapped to logic states '0' and '1', respectively. Considering these states, Boolean logic operations can be performed by arranging the memristive devices into crossbar connections and applying different voltages across them. Numerous models have been proposed in the literature to characterize the memristive devices for SPICE simulation. VTEAM model [10] is one of the models that can characterize various memristive technologies and is used for this work.

B. MAGIC Design Style

MAGIC is a stateful logic technique that utilizes memristive devices to implement logic operations, where the resistive states of memristors store the inputs and outputs of these operations. The MAGIC design style incorporates NOR and NOT gate implementations. To perform MAGIC operations, an initialization step is required, initializing the output memristor to logic '1' before executing the operations. During a NOR operation, an execution voltage (V_{in}) is applied to the input memristors (A, B), while the output memristor is connected to the ground, as illustrated in Fig. 1 ❹. In contrast, the NOT operation only requires two memristors. Since the NOR gate serves as a universal gate, any logic function can be achieved by combining these gates sequentially. The initial step involves converting the given Verilog design netlist into MAGIC NOR and NOT netlists. Fig. 1 showcases the mapping of the netlist given in ❶ to ❷, which is done using the SIMPLE mapping tool.

C. SIMPLER Mapping Tool

SIMPLER MAGIC is a synthesis and mapping tool that is used to generate the MAGIC design style-based mapping of any arbitrary design [6]. The SIMPLER tool generates the mapping of the design to one row of a memristor crossbar. The work shows that these designs can then be replicated across multiple rows of the crossbar, which can operate on different data. Another advantage of doing the same is that the controller and additional circuits cost can be amortized as they can be shared across the multiple rows of the crossbar as they are performing the same set of operations. Hence the mapping obtained using SIMPLER is inherently suitable for *single instruction multiple data* (SIMD) instructions as shown in Fig. 2. Since it supports SIMD-based operations it is useful for applications that require higher throughput. It also supports reusing the same memristors cells to map a larger design to a crossbar of limited size.

D. Related Work

The automated flow to generate for analog implementation has been presented in the literature [11]. The automation is achieved through Cadence skill programming, making it suitable for specific applications. Additionally, the work in [12] automates the attachment of peripherals to the RRAM memory, though limited to memory functionality. Various simulators have been proposed at different levels of abstraction, including system, architecture, circuit, and device levels [13]. Some approaches have also explored mixed-signal simulation, integrating different levels together [14]. However, these existing approaches are often limited to a single device type and fixed parameters. In contrast, our paper introduces a digital LiM implementation, expanding the usability of RRAM beyond conventional memory and analog applications. The proposed implementation allows for configurable parameters, providing flexibility in simulations and enabling more extensive exploration of LiM designs.

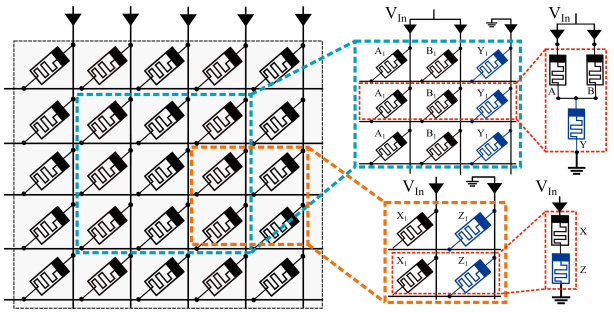


Fig. 2. Demonstrating the simultaneous execution of MAGIC NOT and NOT gates in parallel SIMD fashion. The row-wise parallel approach depicted in the Figure.

Moving toward energy consumption techniques, the current methods for calculating energy consumption involve multiplying the *average energy used during an operation by the number of such operations* in an application, which is a highly coarse-grained approach to determine the energy consumed by the MAGIC design style [7]. Surprisingly, despite its popularity, this methodology falls short of providing accurate estimates of the energy dissipated by an application since it does not account for the energy consumed during initialization, reading, and loading input patterns. As far as we know, no existing framework has the capability to conduct an in-depth analysis of energy consumption in LiM. This paper addresses this gap by introducing the MemSPICE framework, which enables detailed energy consumption analysis for LiM designs by running a SPICE-level simulation.

III. MemSPICE Framework

This section presents the MemSPICE framework to obtain accurate energy estimates by mapping the Verilog design to the MAGIC design style at the SPICE level. The MemSPICE methodology is shown in Fig 3. It is an automated framework to generate the SPICE-level netlist for MAGIC NOR and NOT gate. The automated process comprises three-step (A) SIMPLER tool mapping, (B) automated SPICE-level netlist, (C) MemSPICE output & test bench creation, and (D) energy estimations. In the following section, we discuss each step in detail.

A. Verilog Synthesis

The process starts with the Verilog design synthesis using the ABC synthesis tool [15]. As an illustration, we consider a half-adder example, and you can find the Verilog declaration in Listing 1. This Verilog file (Fig. 3 ❶) serves as an input, alongside the MAGIC cell library (NOT and NOR gates) shown in Fig. 3 ❷, to the ABC tool. Subsequently, the ABC tool synthesizes (Fig. 3 ❸) the Verilog description into NOR and NOT gates. The resulting NOR/NOT netlist becomes the input for the SIMPLER mapping tool (Fig. 3 ❹), which then generates an optimal mapping for MAGIC design style gates. Additionally, the SIMPLER mapping tool performs a sequential mapping of the MAGIC NOT and NOR operations, utilizing three and two memristors on the crossbar, respectively. Crucial information such as the required number of cycles, input/output memristors, and

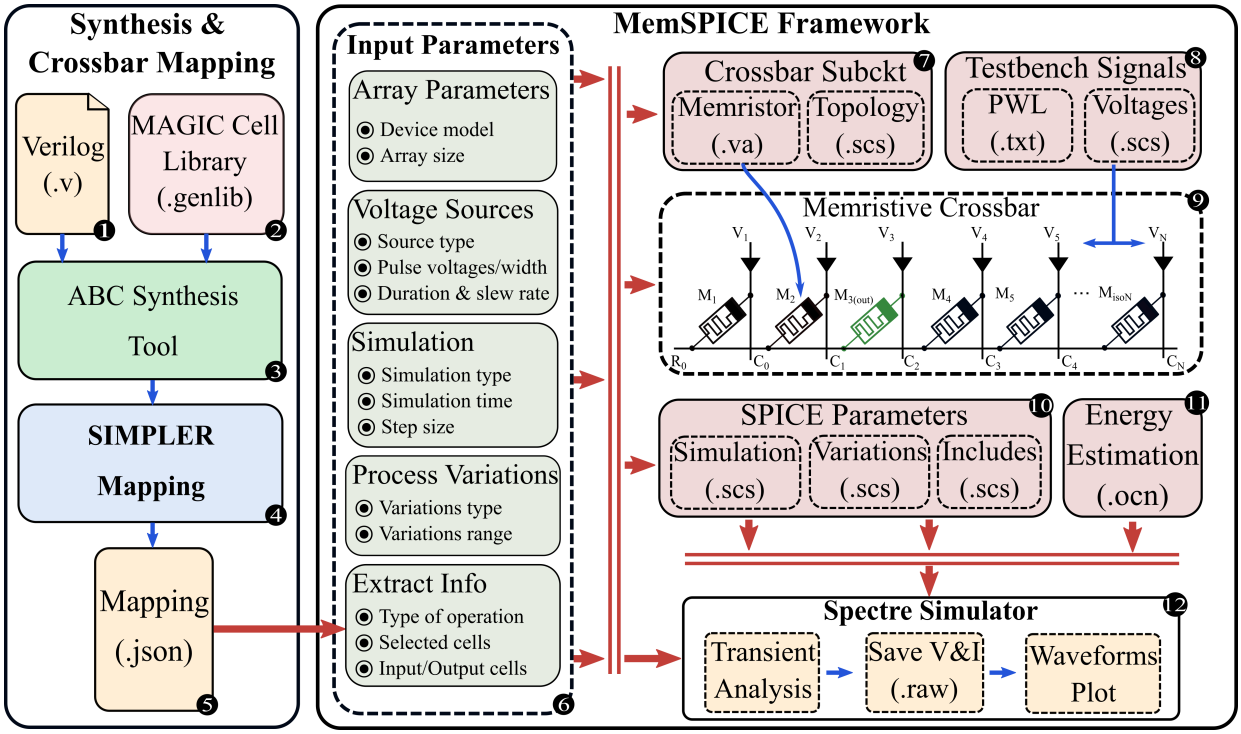


Fig. 3. MemSPICE, an automated digital synthesis framework for LiM computing. It has two major blocks, the first to synthesize the Verilog design on possible logic gated on the crossbar, and the second block automatically generates the SPICE-level simulation files.

other relevant details tailored to the specific application or benchmark is generated by the SIMPLER tool, and this data is stored in a .json file (Fig. 3 ❺). Listing 2 shows the .json file representing the mapping of the half-adder on five memristors connected in a single row. Further, this .json file is used to generate the SPICE-level netlist and test vectors for simulation and detailed energy analysis.

B. Automated SPICE Netlist

MemSPICE utilizes the output from the SIMPLER tool in .json format as input to automatically generate the SPICE-level netlist. From this *.json* file, MemSPICE extracts critical information such as input and output data devices and the sequence of NOR/NOT operations for execution. Additionally, MemSPICE takes several parameters as input to customize the simulation process, which is presented in Fig. 3 ❻. These parameters are organized into four categories based on their role during the simulation:

i) Array parameters: These encompass device model and array size arguments essential for the simulation. As the SIMPLER tool maps the design to a single row, the generated SPICE-level netlist also contains a single row with 'n' numbers of columns.
ii) Voltage-related parameters: MemSPICE supports pulse and piece-wise linear (PWL) voltage sources and allows control over parameters such as pulse amplitude, width, period, and rise/fall time of the voltage source for testbench purposes.
iii) Simulation-related parameters: These parameters offer flexibility in choosing the type of simulation (DC simulation, transient) required for testing, along with defining simulation time and step size during the simulation.
iv) Process variations: This set of parameters controls the type and range of variation for the devices. When process variation parameters are set, MemSPICE draws variation values from a normal distribution probability function around the mean value, with a defined standard deviation in input parameters.

By effectively managing these input parameters, MemSPICE empowers users to tailor the SPICE-level netlist generation and simulation according to their specific requirements, enhancing the flexibility and accuracy of the overall analysis. Considering all the parameters, MemSPICE formulates a spectre-compatible netlist (.scs) in a crossbar configuration. Subsequently, this crossbar SPICE netlist is assembled into a crossbar symbol, complete with dedicated inputs and outputs optimized for benchmarking purposes. In the output, MemSPICE generates multiple .scs files, encompassing the crossbar sub-circuit, testbench signals, as well as simulation and energy estimation files, all essential for conducting SPICE simulations.

```
module half_adder (A, B, S, Cout);
//Input Ports Declarations
input A, B;
//Output Ports Declarations
output S, Cy;
//Logic
assign S = A ^ B ;
assign Cy = (A & B);
endmodule
```

Listing 1. Verilog logic of half adder as an input to MemSPICE.

```
"Row size": 5,
"Number of Gates": 5,
"Inputs": "{A(0),B(1)}",
"Outputs": "{S(4),Cy(2)}",
"Reuse cycles": 1,
"Execution sequence": {
"T0": "Init{'D(2)','D(3)','D(4)'}",
"T1": "n5_(4)=inv1{A(0)}",
"T2": "n6_(3)=inv1{B(1)}",
"T3": "Cy(2)=nor2{n6_(3),n5_(4)}",
"T4": "Init{n5_(4),n6_(3)}",
"T5": "n8_(3)=nor2{B(1),A(0)}",
"T6": "S(4)=nor2{n8_(3),Cy(2)}"}
```

Listing 2. Mapping of a half adder onto five memristors in a row.

C. MemSPICE Output & Test-benching

MemSPICE generates various files in the output, all compatible with spectre simulation. These files are individually created and called in a single main file to execute the final simulation.

1) Crossbar Subckt: MemSPICE first considers a device model (.va) and creates a symbol based on it. Crossbar Subckt block as shown in Fig. 3 ❼ connects symbols in crossbar topology as defined by the input arguments. To adhere to the MAGIC design style's requirement of switches at each row and column, MemSPICE automatically attaches switches at the crossbar pins for convenient access. An example of the crossbar sub-circuit file for the implementation of a half-adder is depicted in Listing 3. In this listing, the crossbar comprises six pins (r_7 and c_0-c5), with a total of six devices connected between these pins. Additionally, seven switches are connected to memristor pin a .scs file containing this information is exported as shown in Listing 4.

```
\\ Connect memirstors together in crossbar
subckt crossbar_sub r0   c0 c1 c2 c3 c4 c5
I0 (r0 c0 n0)  VTEAM_model <model parameters>
I1 (r0 c1 n1)  VTEAM_model <model parameters>
---
ends crossbar_sub
\\ call sub circuit for pheripherls
crossbar0 (sub0_r0  sub0_c0 sub0_c1 sub0_c2
sub0_c3 sub0_c4 sub0_c5) crossbar_sub
```

Listing 3. Automated crossbar sub circuit for half adder implementation.

```
\\ Relay for modelling transistor as switch
W0 (0 sub0_r0 v_r0 0) relay ropen=1T rclosed=1.0
W1 (v_c0 sub0_c0 v_s0 0) relay ropen=1T rclosed=1.0
W2 ---
---
```

Listing 4. Switch connected to rows and columns of the crossbar subckt.

2) Testbench Signals: The testbench signals encompass voltage sources and their timing values, which are dependent on the execution sequence and the type of operation within that sequence. These voltages are crucial for performing operations and initializing or loading input data to the devices. Additionally, the selection of the device for each operation is determined by the voltages on switches, extracted from the execution sequence within the .json file. As the operations on the devices are highly flexible, Fig. 3 ❽ takes all these variations into consideration and generates voltage sources for the crossbar pins along with the PWL file as shown in Listing 5. The PWL file contains voltage waveforms in text format, essential for ensuring the functional correctness of the operations.

```
\\ Create voltage source with PWL file path
V0 (v_r0 0) vsource type=pwl file = "./pwl/r0.txt"
V1 (v_c0 0) vsource type=pwl file = "./pwl/c0.txt"
---
V7 (v_s0 0) vsource type=pwl file = "./pwl/s0.txt"
---
```

Listing 5. Voltage source connected to rows and columns of the crossbar for operation, the actual value of voltage amplitude and duration are defined in PWL files.

To achieve digital LiM using memristors, five different voltage sources are required, each tailored to specific aspects of the operation and ensuring the proper functioning of MAGIC operation.

- ***Input Voltage:*** The input voltage determines the resistances of the memristors used as inputs. It is mapped to 2.0V for storing '1' (LRS) and 0.0V for HRS (memristors' default state).
- ***Read Voltage:*** The read voltage is applied to read the state of the output memristors after all operations. It is set at 0.2V, relatively low compared to SET and RESET voltages, ensuring reliable read operations. There is potential for reducing energy consumption by further decreasing the read voltage.
- ***Initialization Voltage:*** The intermediate output memristors, storing intermediate results, are initialized to LRS for accurate operation. This voltage configures them to LRS and is mapped to 2.0V for correct operation.
- ***Switch Voltage:*** During the execution cycle, only selected devices are active, isolated from others using a switch voltage. The switch voltage of the chosen devices is set to 2.0V (ON), while others remain at 0V (OFF), preserving their state during computation.
- ***Operation Voltage:*** During operation, specific voltages are applied to memristors for NOR and NOT operations in MAGIC design style. Input memristors are connected to 1.0V during NOT operation, while the output memristor is connected to the ground.

3) SPICE Simulation Parameters: Fig. 3 ❿ is responsible for managing the parameters essential for the simulation tool. These parameters encompass the path of the model and other peripherals within the circuit. Additionally, if there are any variations to be considered during the simulation, this block handles the variation values which are mapped to the devices during simulation. It also maps the input pattern to the corresponding device state. Moreover, it includes simulation analysis parameters that dictate how the simulation will be conducted. MemSPICE automatically calculates the simulation time based on the total execution cycles and the duration of pulses for each operation. However, other parameters, such as step size and include path, are analyzed from the input parameters and then parsed into the simulation format.

As depicted in Listing 6, this step involves including the path of the memristor model and specifying parameters like input pattern and variations, if applicable. Additionally, the current values for each memristor are saved during the simulation, and these values are later utilized for energy analysis. By following this process, the parameters are effectively integrated and employed during the simulation, thereby enhancing the accuracy and efficiency of the overall analysis.

```
\\ include memristor verilog model
ahdl_include "<path_to>/verilog.va"
parameters in0=0 in1=2 <input & variation parameters>
\\ save the current for futher analysis
save crossbar0.I0:n ---
tran tran stop=2.4e-08 step=1e-12 maxstep=1e-12
```

Listing 6. SPICE simulation parameters.

D. Energy Estimation

To accurately calculate energy consumption, we simulate the circuit (Fig. 3 ⓬) and generate a waveform file. Energy per device is calculated using Equation 1, summing the product of voltage and current over simulation time.

$$\text{Energy} = \sum_{\text{i=0}}^{\text{n}} \int_{0}^{\text{t}} (\text{V}_\text{i} \times \text{I}_\text{i}) \text{dt} \tag{1}$$

'n' denotes memristor count, and 't' represents simulation time based on pulse width and cycles required to complete the benchmark. The equation covers all memristor activity, including initialization, execution, and read energy.

The total energy consumed for a given application is computed by combining Equation 1 for each device within the network. These equations are stored as a .ocn file, as shown in Fig. 3 ⓫. This file serves as a repository for the energy equations, facilitating their systematic application and enabling the determination of the overall energy consumption of the network.

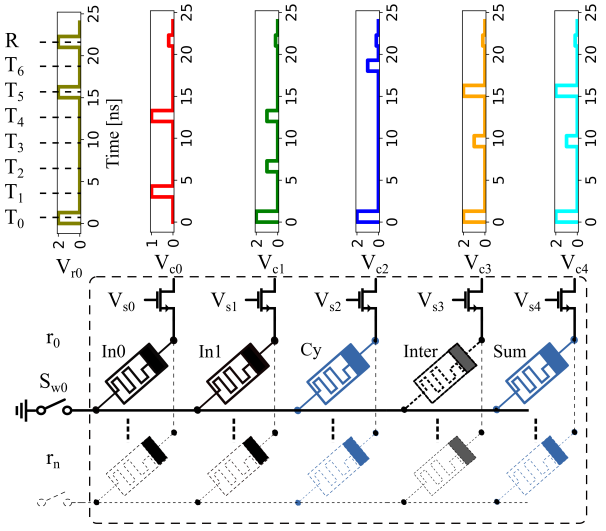


Fig. 4. Half adder implementation for "10" input ($in_0 = 1$, $in_1 = 0$) using five memristors in a row as per the mapping in Listing 2. This implementation requires a total of 7 execution cycles, including an initialization (T_0) and a reused cycle (T_4). Similar to a SIMD architecture, the same operation can be executed concurrently in different rows (r_n).

Fig. 4 presents the schematic and waveform generated for the half-adder implementation, which is synthesized from Listing 1. Notably, the half adder implementation is mapped to a single row of 5 memristors. As shown in Listing 2, it also requires memristor reinitialization to store intermediate results. These intermediate results are represented by the dotted line with memristors labeled as 'Inter', while the final sum and carry are stored in the memristors labeled as 'Sum' and 'Cy', respectively.

The implementation necessitates six voltage sources (V_{r0} and $V_{c0}-V_{c4}$) with values tailored to the logic implementation during each execution sequence. Additionally, there are five other voltage sources responsible for opening switches to execute operations. Furthermore, these operations can be performed in SIMD fashion, but they are beyond the scope of this work.

In summary, the framework offers digital and circuit designers the ability to thoroughly test and optimize their designs in a more accurate and automated manner, providing valuable insights into the effectiveness of their methodologies.

IV. Experimental Results

This section presents the results achieved through the MemSPICE framework. To validate the methodology, we conducted tests using the ISCAS'85 benchmark with the proposed framework. To ensure comparability with existing literature [8], we mapped all benchmarks to a single row of 512 devices. For operating, programming, and reading the state of memristors, we employed different voltage with a 1.3 ns pulse width, with 1 ps rise and fall times.

Table I presents ISCAS'85 benchmark results. It includes benchmark names, primary inputs/outputs count, cycles needed for the final result, NOR/NOT gate counts, re-initialization cycle, and energy values from the literature. MemSPICE allows testing with input patterns for energy consumption. I1, I2, and I3 represent all 0's, all 1's, and alternating 1's and 0's, respectively. We now discuss the results in detail using some examples from the benchmarks.

A. c432 from ISCAS 85

We mapped the c432 benchmark circuit (27-channel interrupt controller) on 512 devices, featuring 36 inputs and 7 outputs. Its operation requires 250 cycles, with 101 NOT operations and 148 NOR operations. The c432 is relatively a small benchmark that easily fits within 512 devices without any reinitialization cycles. The execution energy closely matches the literature, but the initialization energy is considerably higher. As a first step, MemSPICE initializes all devices to '1' (LRS) except for input memristors, which are necessary for MAGIC implementation. In this case, initializing 476 devices (512-PI) to '1' dominates the initialization energy. The execution energy varies significantly based on the input pattern due to the benchmark's smaller size. This is true for all the benchmark circuits that can be mapped to far fewer memristors than 512. The unused memristor energy dominates the operation energy in this case.

Fig. 5 presents the energy breakdown of the c432 benchmark. Since c432 does not have any re-initialization cycle, all the devices are initialized simultaneously in the first cycle for use in the operation cycle. A closer view of Fig. 5(a) is provided in 5(b) and 5(c) to illustrate the execution and read energy, respectively. The read energy is computed by reading the state of each memristor in a row to verify functionality and switching during operation.

B. c3540 from ISCAS 85

As shown in Table I, the c3540 benchmark, which is an 8-bit arithmetic and logic unit, requires 1397 cycles with 50 input and 22 output memristors. It comprises 465 NOT gates and 928 NOR gates. With the circuit mapped to 512 devices, three re-initialization of devices are needed to complete the entire operation, making re-initialization energy a significant factor. The energy consumption using the MemSPICE framework for I1, I2, and I3 input patterns is found to be 2676 pJ, 2474 pJ, and 2431 pJ, respectively. During re-initialization, we apply a SET voltage (2.0V) to reset the device to LRS state without reading its current state. If the device is already in the LRS state before re-initialization, it draws a considerable amount of current, significantly increasing the energy consumption. In this specific case, the re-initialization energy is approximately 70 $\times$ higher than the execution energy.

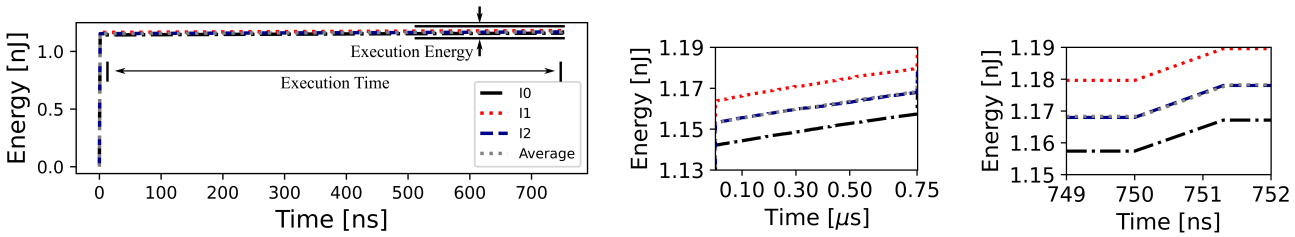
Fig. 6 displays the energy breakdown of the c3540 benchmark. As indicated in Table I, c3540 requires three initialization cycles. The graph in Fig. 6(a) clearly illustrates the significant increase in energy during the re-initialization cycle. This graph highlights that the re-initialization energy dominates the energy consumption during execution. Fig. 6(b) and Fig. 6(c) provide a closer view of Fig. 6(a).

Table I presents the results for all the benchmarks, showcasing how the MemSPICE framework simplifies netlist generation and energy evaluation for MAGIC design style-based circuits. The execution energy values obtained by MemSPICE align closely with state-of-the-art energy calculations, validating the correctness of the methodology. Furthermore, MemSPICE can capture additional energy aspects not addressed in the existing literature. The utilization of SPICE-level simulation [8] in MemSPICE contributes to more precise energy values, emphasizing the framework's significance for accurate energy analysis and automation in digital LiM.

The output of the framework includes waveforms illustrating the executed operations in the sequence, with an explicit read pulse incorporated. To validate the functional correctness of the benchmarks, manual testing is performed, ensuring the accuracy of the files generated by the proposed framework.

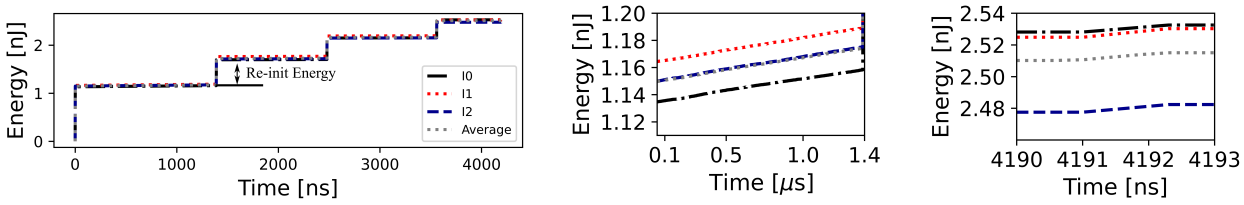
TABLE I
ENERGY CONSUMPTION RESULTS ON ISCAS'85 BENCHMARKS

Circuit	PI/PO	Cycles	NOT	NOR	Reinit	Energy (pJ) Literature [7], [16]	**I1:** Energy (pJ), Input; all 0			**I2:** Energy (pJ), Input; all 1			**I3:** Energy (pJ), Input; alt.		
							Exe	Init	Total	Exe	Init	Total	Exe	Init	Total
c17	5/2	14	7	6	0	0.655	0.674	1161	1161.67	0.90	1164	1164.9	0.75	1162	1162.75
c432	36/7	250	101	148	0	12.31	15.23	1142	1157.23	15.96	1163	1178.96	14.92	1153	1169.92
c499	41/32	605	213	390	1	29.6	30.85	1790	1820.85	25.53	1803	1828.53	28.45	1788	1816.45
c880	60/26	505	194	310	1	24.85	22.91	1819	1841.91	25.26	1785	1810.26	24.76	1779	1803.76
c1908	33/25	571	210	359	1	27.99	26.33	1850	1876.33	22.95	1814	1836.95	25.72	1822	1847.72
c3540	50/22	1397	465	928	3	68.18	37.92	2676	2713.92	37.51	2474	2511.51	38.22	2431	2469.22



(a) (b) (c)

Fig. 5. Energy consumption of c432 benchmark (a) total energy, (b) execution energy and (b) read energy.



(a) (b) (c)

Fig. 6. Energy consumption of c3540 benchmark (a) total energy, (b) execution energy before first reintilization and (b) read energy.

V. CONCLUSIONS

This paper has introduced MemSPICE, a SPICE-level framework that fills the gap in existing research by automating the generation of SPICE netlists for digital LiM using memristors. MemSPICE offers users the ability to control and fine-tune various parameters, providing remarkable flexibility to accommodate diverse configurations and scenarios. It takes Verilog-defined logic and automatically generates SPICE-level simulations for detailed analysis. Notably, it provides precise and fine-grained energy values, enhancing the accuracy of energy estimation. The energy values generated by MemSPICE were found to be in agreement with the energy calculations reported in the literature. MemSPICE will enable researchers, without any circuit knowledge, to accurately estimate the benefits of their proposed ideas by performing automated SPICE-level simulations and we believe this will have a huge impact in this domain of research.

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, German Research Foundation (DFG) within the Project PLiM (DR 287/35-1, DR 287/35-2) and through Dr. Suhas Pai Donation Fund at IIT Bombay.

REFERENCES

[1] S. Kvatinsky *et al.*, "Memristor-based material implication (IMPLY) logic: Design principles and methodologies," *IEEE TVLSI*, vol. 22, no. 10, pp. 2054–2066, 2013.

[2] S. Gupta *et al.*, "Felix: Fast and energy-efficient logic in memory," in *2018 IEEE/ACM ICCAD*. IEEE, 2018, pp. 1–7.

[3] A. Deb *et al.*, "Automated Equivalence Checking Method for Majority based In-Memory Computing on ReRAM Crossbars," in *2023 ASP-DAC*, Jan. 2023, pp. 19–25.

[4] S. Kvatinsky *et al.*, "MAGIC—Memristor-Aided Logic," *IEEE TCAS-II: Express Briefs*, vol. 61, no. 11, pp. 895–899, Nov. 2014.

[5] A. Eliahu *et al.*, *mMPU: Building a Memristor-based General-purpose In-memory Computation Architecture*, 04 2021, pp. 119–131.

[6] R. Ben-Hur *et al.*, "SIMPLER MAGIC: Synthesis and mapping of in-memory logic executed in a single row to improve throughput," *IEEE TCAD*, vol. 39, no. 10, pp. 2434–2447, 2020.

[7] P. L. Thangkhiew *et al.*, "Efficient mapping of boolean functions to memristor crossbar using MAGIC NOR gates," *IEEE TCAS I: Regular Papers*, vol. 65, no. 8, pp. 2466–2476, 2018.

[8] S. Singh *et al.*, "Should we even optimize for execution energy? rethinking mapping for magic design style," arxiv 2023 (accepted).

[9] A. Sebastian *et al.*, "Memory devices and applications for in-memory computing," *Nature nanotechnology*, vol. 15, no. 7, pp. 529–544, 2020.

[10] S. Kvatinsky *et al.*, "VTEAM: A general model for voltage-controlled memristors," *IEEE TCAS II: Express Briefs*, vol. 62, no. 8, pp. 786–790, 2015.

[11] D. Antoniadis *et al.*, "An open-source rram compiler," in *2022 20th IEEE Interregional NEWCAS Conference (NEWCAS)*, 2022, pp. 465–469.

[12] D. D. Antoniadis *et al.*, "Open-source memory compiler for automatic rram generation and verification," in *2021 WSCAS*, 2021, pp. 97–100.

[13] F. Staudigl *et al.*, "A survey of neuromorphic computing-in-memory: Architectures, simulators, and security," *DATE*, vol. 39, no. 2, pp. 90–99, 2022.

[14] M. Fritscher *et al.*, "Simulating memristive systems in mixed-signal mode using commercial design tools," in *26th ICECS*, 2019, pp. 225–228.

[15] A. Mishchenko *et al.*, "Abc: A system for sequential synthesis and verification," *URL http://www.eecs.berkeley.edu/alanmi/abc*, vol. 17, 2007.

[16] C. K. Jha *et al.*, "Imagin: Library of imply and magic nor-based approximate adders for in-memory computing," *IEEE JXCDC*, vol. 8, no. 2, pp. 68–76, 2022.