

Should We Even Optimize for Execution Energy? Rethinking Mapping for MAGIC Design Style

Simranjeet Singh¹, Chandan Kumar Jha², Ankit Bende³, Phrangboklang Lyngton Thangkhiew⁴,
Vikas Rana⁵, Sachin Patkar, Rolf Drechsler⁶, and Farhad Merchant⁷

Abstract—Memristor-based logic-in-memory (LiM) has become popular as a means to overcome the von Neumann bottleneck in traditional data-intensive computing. Recently, the memristor-aided logic (MAGIC) design style has gained immense traction for LiM due to its simplicity. However, understanding the energy distribution during the design of logic operations within the memristive memory is crucial in assessing such an implementation's significance. The current energy estimation methods rely on coarse-grained techniques, which underestimate the energy consumption of MAGIC-styled operations performed on a memristor crossbar. To address this issue, we analyze the energy breakdown in MAGIC operations and propose a solution that utilizes mapping from the SIMPLER MAGIC tool to achieve accurate energy estimation through SPICE simulations. In contrast to existing research that primarily focuses on optimizing execution energy, our findings reveal that the memristor's initialization energy in the MAGIC design style is, on average, $68\times$ higher. We demonstrate that this initialization energy significantly dominates the overall energy consumption. By highlighting this aspect, we aim to redirect the attention of designers towards developing algorithms and strategies that prioritize optimizations in initializations rather than execution for more effective energy savings.

I. INTRODUCTION

COMPUTING-IN-MEMORY is one way of reducing the impact of the von Neumann bottleneck. As a result, digital logic-in-memory (LiM) has gained significant momentum recently. Memristive memories are seen as a viable candidate for LiM. Memristors possess two distinctive states: the high resistive state (HRS) and the low resistive state (LRS), which are then mapped to boolean logic '0' and logic '1', respectively. These resistive states are the foundation for designing Boolean logic gates within the memory. To model the memristive behavior in SPICE, various model has been proposed in the literature. The VTEAM model [1] is derived from the derivative of the internal state variables. This model

This work was supported in part by the BMBF, Germany, in the project NEUROTEC II under Project 16ME0398K, Project 16ME0399, DFG within the Project PLiM (DR 287/35-1, DR 287/35-2) and through Dr. Suhas Pai Donation Fund at IIT Bombay.

Simranjeet Singh and Sachin Patkar are with the IIT Bombay, Mumbai 400076, India, and Simranjeet Singh is also with Forschungszentrum Jülich GmbH, PGI, Jülich 52425, Germany (e-mails: {simranjeet, patkar}@ee.iitb.ac.in).

Ankit Bende and Vikas Rana are with Forschungszentrum Jülich GmbH, PGI, Jülich 52425, Germany (e-mails: {a.bende,v.rana}@fz-juelich.de).

Chandan Kumar Jha and Rolf Drechsler are with the Institute of Computer Science, University of Bremen, Bremen 28359, Germany. Rolf Drechsler is also with the Department of Cyber-Physical Systems, DFKI GmbH, Bremen 28359, Germany (e-mail: {chajha, drechsler}@uni-bremen.de).

Phrangboklang Lyngton Thangkhiew is with the IIIT Guwahati, Assam 781015, India (e-mail: phrangboklang.thangkhiew@iiitg.ac.in).

Farhad Merchant is with Newcastle University, Newcastle upon Tyne NE17RU, United Kingdom (e-mail: farhad.merchant@newcastle.ac.uk).

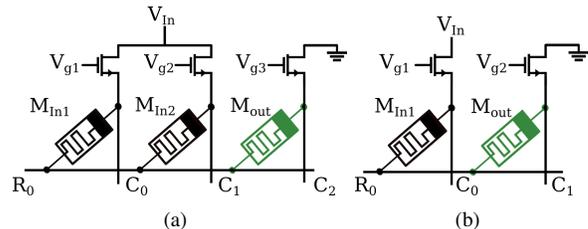


Fig. 1. MAGIC design style based (a) NOR gate, and (b) NOT gate

can accurately represent the switching characteristics observed in memristor devices, making it a suitable choice for this study.

Numerous approaches have been proposed in the literature to implement logic gates using memristors, including techniques such as memristor-aided logic (MAGIC) [2], memristor-based material implication (IMPLY) [3], fast and energy-efficient logic in memory (FELIX) [4], and majority logic [5]. MAGIC has emerged as a widely adopted technique among these approaches due to its superior energy efficiency and latency performance [6].

MAGIC is a stateful logic technique to implement logic operations using memristive devices, where inputs and outputs of logic operations are stored in the resistive states of memristors. Fig. 1 visually represents the implementation of MAGIC NOR (1a) and NOT (1b) gates using memristors. An output memristor, M_{out} , is initialized to LRS, and it changes the state from LRS to HRS based on the input state stored in M_{in1} and M_{in2} . NOR and NOT operations use three (two input, one output) and two memristors (one input, one output) in a single operation, respectively. The output of these logic gates can be conveniently stored in a dedicated output memristor without requiring any specific arrangement in a crossbar, making this technique particularly suitable for digital LiM applications. The state-of-the-art tool in this field, known as the SIMPLER MAGIC [7], is designed explicitly for synthesizing the MAGIC NOT and NOR operations into a single-row memristor crossbar. Henceforth, we refer to SIMPLER MAGIC as SIMPLER.

The utilization of the MAGIC design style and its mapping onto the crossbar has been suggested for creating an in-memory general-purpose processing unit, mMPU [6]. As this design style is rapidly gaining popularity in mainstream computing, assessing the amount of energy consumed by this technique is crucial. 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

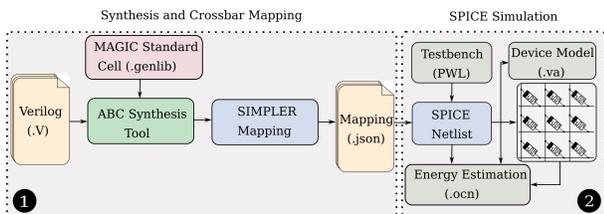


Fig. 2. Methodology for generating a SPICE-level netlist and estimating energy for any given workload.

style [8]. 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.

This letter shows a novel and accurate *fine-grained* methodology for energy calculation of LiM, where we break down the energy consumption for the MAGIC design style. We generate a SPICE-level netlist and testbench voltages for a given application to find the accurate energy breakdown. 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.

- Firstly, a fine-grained methodology for energy calculation for MAGIC design style. (Section II)
- A detailed evaluation for energy consumption where each device's consumption is measured in a crossbar, irrespective of its contribution to the operation. (Section III)
- Lastly, The discussion on approx. 70x average energy gap for ISCAS'85 benchmarks when compared to the state-of-the-art energy calculation methods. (Section IV and V)

II. METHODOLOGY

Overall energy calculation methodology comprises three steps (see Fig.2): i) synthesis and crossbar mapping, ii) SPICE netlist generation, and iii) testbench and energy and estimation.

A. Synthesis and Crossbar Mapping

We begin the process by synthesizing the Verilog design using the ABC synthesis tool [9]. The ABC tool allows to synthesis of any arbitrary logic function into NOR and NOT logic gates as shown in Fig. 2 ①. Subsequently, the NOR/NOT netlist serves as input for the SIMPLER mapping tool, generating an optimal mapping for MAGIC design style gates. Moreover, the SIMPLER mapping tool performs a sequential mapping of the MAGIC NOT and NOR operations, which require the utilization of three and two memristors on the crossbar, respectively. Additionally, the SIMPLER tool generates the necessary information, such as the required number of cycles, input/output memristors, and other relevant details specific to the application or benchmark, which are then stored in a .json file. Listing 1 shows the .json of half adder mapping on five memristors connected in a single row. Moving forward, the .json file has been used to generate the SPICE-level netlist and test vectors.

B. SPICE Netlist Generation

As shown in Fig. 2 ②, SPICE simulation takes input from the synthesis and mapping block to generate the SPICE-level netlist. In the SPICE simulation block, the test bench containing voltage files and energy estimation scripts is also

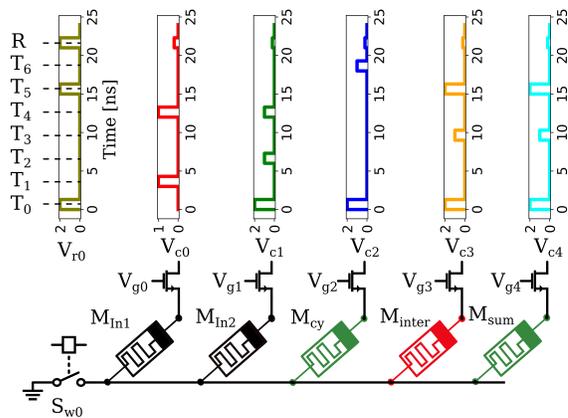


Fig. 3. Implementation of a half adder using five memristors arranged in a row, based on the mappings provided in Listing 1. The waveform configuration involves a total of 7 execution cycles (T_0 - T_6), including an initialization cycle (T_0), a reused cycle (T_4), and a final read cycle ('R') to observe the output states of each device.

created for accurate energy estimation. The critical information is extracted from the crossbar mapping (.json) file. This information includes the input and output data devices and the execution sequences of NOR/NOT operations. Based on the required number of memristors for the given set of applications, the crossbar netlist is designed using the VTEAM model as memristor devices. All operations are mapped to a single row of the crossbar, where 'n' memristors are connected to columns. A controllable switch connects the row of the crossbar to the ground during the writing process while it remains floating as a requirement for MAGIC execution. Considering all the relevant parameters, a spectre-compatible netlist (.scs) is designed in the structure of a crossbar. Subsequently, the crossbar SPICE netlist is encapsulated within a crossbar symbol with dedicated input and output for benchmarking purposes.

```

"Row size": 5,
"Number of Gates": 5,
"Inputs": "{A(0), B(1)}",
"Outputs": "{S(4), Cout(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": "Cout(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), Cout(2)}"
}

```

Listing 1. Mapping of a half adder onto five memristors in a row.

C. Testbench and Energy Estimation

To enable the operations in digital in-memory computing, input voltages play a crucial role in configuring the functionality. Executing any arbitrary digital logic on the memristor crossbar becomes possible by applying different voltage combinations with specific values to the rows and columns. In digital LiM using 2 memristors, there are five distinct voltage levels required:

- **Input Voltage:** This voltage determines the resistances of the memristors used as inputs. The input voltage is mapped to 2.0V for storing '1' (LRS) and 0.0V for HRS (by default, the memristors are in HRS).

- **Read Voltage:** Applied to read the state of the output memristors after all operations have been performed. The read voltage is relatively low, set at 0.2V in this case, compared to the SET and RESET voltages. The read voltage is chosen to ensure reliable read operations; however, there is potential for reducing energy consumption by reducing the read voltage.
- **Initialization Voltage:** The intermediate output memristors, which store intermediate results, need to be initialized to LRS before accurate operation. This voltage configures the intermediate output memristors to LRS and is mapped to 2.0V for correct operation.
- **Gate Voltage:** During the execution cycle, only a few devices are chosen for operation, while others in the crossbar must be isolated. The gate voltage is used to select the devices for operation. The gate voltage of the selected devices is set to 2.0V (ON), while the others are kept at 0V (OFF). This voltage ensures that the state of these memristors remains unaltered during computation.
- **Operation Voltage:** During operation, a specific voltage is applied to the memristors with input values to execute MAGIC design style-based NOR and NOT operations. A voltage of 1.0V is applied to one memristor during the NOT operation and to two memristors during the NOR operation. The intermediate output memristor is connected to the ground.

The implementation of these voltage values involves the use of a piece-wise linear voltage source. The determination of specific voltage levels is based on a single execution cycle provided in the .json file. Fig. 3 demonstrates the voltage waveform explicitly designed for implementing an in-memory half adder, as depicted in Listing 1. The SPICE-level netlist, which incorporates the VTEAM memristor model (.va) and the corresponding voltage sources, is compatible with Cadence spectre simulation.

Energy Estimation: To accurately calculate the energy consumption, we perform a simulation for the required duration and generate a waveform file. The energy is calculated regardless of the device’s selection. The total energy is determined using $\sum_{i=0}^n \int_0^t (V_i \times I_i) dt$, which sums up the product of voltage and current over simulation time. Here, ‘n’

represents the number of memristors, and ‘t’ corresponds to the simulation time. The simulation time depends on the pulse width and the total number of cycles necessary to complete the benchmark. The given equation captures the activity on each memristor irrespective of its use in the cycle, which includes initialization, execution, and read energy.

This section provided a detailed discussion on netlist generation, testbench generation, and energy estimation techniques for a specific benchmark. Furthermore, the simulated design is evaluated using the described methodology to assess its energy efficiency, and the results are further compared with state-of-the-art methodologies.

III. EXPERIMENTAL RESULTS

The state-of-the-art methods used to calculate energy consumption in MAGIC design style applications rely on a coarse-grained approach, multiplying the average energy per operation by the total number of operations in the application. As discussed in [8], this approach does not accurately capture the energy consumed by the MAGIC design style. Additionally, existing literature often calculates energy by applying a DC source for execution and measuring energy at the exact switching point. However, a pulse is required for operation in real implementations. In Table I, we compare the energy values obtained from the literature with the energy values calculated using our methodology. The results show that the execution energy for both NOT and NOR implementations closely matches the reported energy values in the literature [10].

In the MAGIC design style, energy consumption is predominantly dominated by the writing phase. Table I demonstrates that the writing process consumes $16.8\times$ more energy during SET operation than the reported energy values, considering a pulse width of 1.3 ns and rise/fall times of 1 ps. Our proposed methodology uses a pulse width of 1.3 ns for writing since it is also the minimum required pulse width for correct operations. We want to highlight that energy consumption shoots up significantly if a larger pulse width is used. Due to the memristor’s low resistance, it draws a significant current. To simplify the design of peripherals, a single pulse width is utilized, determined based on the worst-case pulse requirement in design.

The overall energy results are shown in Table II. The first and second columns have the benchmark suite’s names and their respective benchmark circuits, respectively. The PI/PO column gives the number of primary inputs and primary outputs. The ‘Cycles’ column gives the number of cycles required to obtain the final result for the given benchmark circuit. The ‘NOR’ and the ‘NOT’ columns give the number of the NOR and NOT operations, respectively. The energy consumption using the current state-of-the-art method is shown in the next column. In the last three columns, we show the energy consumption of the various benchmark circuit using three different input patterns, P1, P2, and P3, respectively. The pattern P1 denotes all 0’s at the input, the pattern P2 denotes all 1’s at the input, and the pattern P3 denotes alternating 1’s and 0’s at the input. We now discuss the results in detail using some examples from the benchmarks.

Fig. 4 illustrates the energy breakdown for the c3540 benchmark, with a focus on c3540 due to its larger size, which

TABLE I
MAGIC NOR, NOT AND WRITE ENERGY ESTIMATION

| NOR, NOT& Write Operation | $V_{in} = 1.0@1.3$ ns DC source | $V_{in} = 1.0@1.3$ ns rise and fall time (1 ps) |
|------------------------------|---------------------------------|---|
| NOR | Energy (fJ) | Energy (fJ) |
| 00 \rightarrow 1 | 8.6 | 8.6 |
| 01 \rightarrow 0 | 89.53 | 87.96 |
| 10 \rightarrow 0 | 89.53 | 87.96 |
| 11 \rightarrow 0 | 32.48 | 30.48 |
| Average | 55.04 | 53.75 |
| NOT | Energy (fJ) | Energy (fJ) |
| 0 \rightarrow 1 | 4.31 | 4.32 |
| 1 \rightarrow 0 | 90.31 | 88.74 |
| Average | 47.31 | 46.53 |
| Writing | Energy (fJ) @1.0 ns | Energy (fJ) |
| RESET \rightarrow SET (2V) | 75.56 | 1272.2 |
| SET \rightarrow RESET (1V) | 17.66 | 19.01 |

TABLE II
ENERGY CONSUMPTION RESULTS ON ISCAS'85 BENCHMARKS

| Circuit | PI/PO | Cycles | NOT | NOR | Energy (pJ) Literature [8], [10] | P1: Energy (pJ), Input; all 0 | | | P2: Energy (pJ), Input; all 1 | | | P3: Energy (pJ), Input; alt. | | |
|---------|-------|--------|-----|-----|-------------------------------------|-------------------------------|-------|-------|-------------------------------|-------|-------|------------------------------|-------|-------|
| | | | | | | Init | Exe | Read | Init | Exe | Read | Init | Exe | Read |
| c17 | 5/2 | 14 | 7 | 6 | 0.655 | 1161 | 0.674 | 11.55 | 1164 | 0.90 | 11.55 | 1162 | 0.75 | 11.56 |
| c432 | 36/7 | 250 | 101 | 148 | 12.31 | 1142 | 15.23 | 9.73 | 1163 | 15.96 | 9.95 | 1153 | 14.92 | 10.04 |
| c499 | 41/32 | 605 | 213 | 390 | 29.6 | 1790 | 30.85 | 6.23 | 1803 | 25.53 | 7.50 | 1788 | 28.45 | 6.73 |
| c880 | 60/26 | 505 | 194 | 310 | 24.85 | 1819 | 22.91 | 7.25 | 1785 | 25.26 | 8.024 | 1779 | 24.76 | 7.504 |
| c1908 | 33/25 | 571 | 210 | 359 | 27.99 | 1850 | 26.33 | 7.13 | 1814 | 22.95 | 7.701 | 1822 | 25.72 | 7.39 |
| c3540 | 50/22 | 1397 | 465 | 928 | 68.18 | 2676 | 37.92 | 4.41 | 2474 | 37.51 | 5.41 | 2431 | 38.22 | 4.84 |

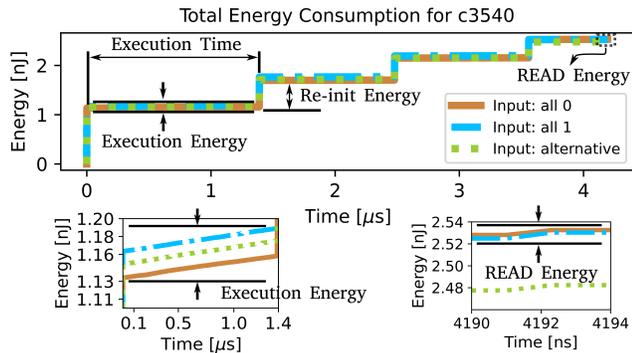


Fig. 4. Energy breakdown and execution time analysis for the c3540 benchmark, highlighting different energy consumption components. The zoomed-in versions of the execution and read energy are highlighted in separate graphs.

requires more initialization compared to other benchmarks. We see that the c3540 benchmark circuit from ISCAS 85 (Table II) has 50 inputs and 22 outputs. The number of cycles required for the operation is 1397. The circuit consists of 465 NOT gates and 928 NOR gates. The current methods that only evaluate the energy of the operation give 68.18 pJ as the overall energy consumption. We obtained the energy consumption with SPICE netlist for P1, P2, and P3 to be 2676 pJ, 2474 pJ, and 2431 pJ, respectively. The difference in the energy consumption value is approx. 40 \times compared to the current state of the art. The difference in energy consumption compared to the state-of-the-art method is lower than the c17 benchmark circuit as the design is larger than the entire crossbar and reuses the crossbar to perform the operations. The difference in the energy consumption between P1, P2, and P3 highlights that the operation energy dominates the initialization energy, and depending upon the input pattern, different energy values will be obtained. This allows us to capture the energy consumption of the design depending on the input patterns.

The results for all the other benchmarks are shown in Table II. The energy values obtained using our methodology are very different than the ones obtained using the state-of-the-art energy calculation methodology. On average, the energy consumption of benchmark c432 to c3540 is 68 \times higher than the energy values presented in the literature.

IV. DISCUSSION

Based on SPICE simulation results, the energy consumed during logic implementation in memory is primarily dominated by the initialization process, typically considered a one-time energy consumption. However, in resource-constrained implementations, the initialization energy becomes the domi-

nant factor. It may be argued that reducing the pulse width towards the switching threshold can lower energy consumption. Nevertheless, even in such cases, the re-initialization energy will still dominate the overall energy consumption. We believe that this insight can be incorporated while developing efficient mapping algorithms, which is missing in the current works.

In conclusion, the initialization energy dominates the energy consumption of a given benchmark. Secondly, the read energy needs to be considered. Contrary to prior works, we saw that execution energy is negligible as compared to initialization and read energy. Since the majority of prior works focus on reducing execution energy, we believe our work will pave the way for further research to optimize mapping strategies that prioritize initialization.

V. CONCLUSION

This work introduces a methodology for accurately calculating the fine-grained energy consumption of logic operations in the MAGIC design style. Through SPICE simulation, we demonstrate that initialization energy significantly dominates the overall energy in MAGIC design style implementation, accounting on average for 68 \times more energy consumption as compared to presented state-of-the-art energy values. These findings emphasize the need for researchers to prioritize the optimization of initialization rather than execution. In future studies, we aim to develop mapping techniques further to optimize energy consumption in the MAGIC design style.

REFERENCES

- [1] S. Kvatinisky *et al.*, "Vteam: A general model for voltage-controlled memristors," *IEEE TCAS II: Express Briefs*, vol. 62, no. 8, pp. 786–790, 2015.
- [2] S. Kvatinisky *et al.*, "MAGIC—Memristor-Aided Logic," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 61, no. 11, pp. 895–899, Nov. 2014.
- [3] J. Borghetti *et al.*, "'Memristive' switches enable 'stateful' logic operations via material implication," *Nature*, vol. 464, no. 7290, pp. 873–876, Apr. 2010.
- [4] S. Gupta *et al.*, "Felix: Fast and energy-efficient logic in memory," in *2018 IEEE/ACM ICCAD*. IEEE, 2018, pp. 1–7.
- [5] 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.
- [6] A. Eliahu *et al.*, *mMPU: Building a Memristor-based General-purpose In-memory Computation Architecture*, 04 2021, pp. 119–131.
- [7] 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.
- [8] 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.
- [9] A. Mishchenko *et al.*, "Abc: A system for sequential synthesis and verification," *URL http://www.eecs.berkeley.edu/alanmi/abc*, vol. 17, 2007.
- [10] C. K. Jha *et al.*, "Imagin: Library of imply and magic nor-based approximate adders for in-memory computing," *IEEE JXDC*, vol. 8, no. 2, pp. 68–76, 2022.