Circuit Topology and Synthesis Flow Co-Design for the Development of Computational ReRAM

Carlos Fernandez
Dept. of Electronic Engineering
Universidad Tecnica Federico Santa Maria (UTFSM)
Valparaiso, Chile
carlos.fernandezh@sansano.usm.cl

Ioannis Vourkas
Dept. of Electronic Engineering
Universidad Tecnica Federico Santa Maria (UTFSM)
Valparaiso, Chile
ioannis.vourkas@usm.cl

Antonio Rubio
Dept. of Electronic Engineering
Universitat Politècnica de Catalunya (UPC)
Barcelona, Spain
antonio.rubio@upc.edu

***Abstract*— Emerging memory technologies will play a decisive role in the quest for more energy-efficient computing systems. Computational ReRAM structures based on resistive switching devices (memristors) have been explored for in-memory computations using the resistance of ReRAM cells for storage and for logic I/O representation. Such approach presents three major challenges: the support for a memristor-oriented logic style, the ad-hoc design of memory array driving circuitry for memory and logic operations, and the development of dedicated synthesis tools to instruct the multi-level operations required for the execution of an arbitrary logic function in memory. This work contributes towards the development of an automated design flow for ReRAM-based computational memories, highlighting some important HW-SW co-design considerations. We briefly present a case study concerning a synthesis flow for a *nonstateful* logic style and the co-design of the underlying 1T1R crossbar array driving circuit. The prototype of the synthesis flow is based on the ABC tool and the Z3 solver. It executes fast owing to the level-by-level mapping of logic gates. Moreover, it delivers a mapping that minimizes the logic function latency through parallel logic operations, while also using the less possible ReRAM cells.**

Keywords— memristor, resistive RAM, in-memory computing, logic synthesis, electronic design automation, ratioed logic

I. Introduction

Emerging nonvolatile memory technologies will play an important role in the current quest for more energy efficient computing systems, driven by the rising amounts of data needed to be processed [1]. Resistive switching devices (memristors) organized in crossbar arrays to form resistive random-access memories (ReRAM), are considered among the key enabling device technologies for *computational memory structures* [2]. Different from the classical Von Neumann approach, such novel system architectures will reduce (or eliminate) the data movement between memory and processing modules. The driving circuitry of the crossbar will allow not only the control and the communication with the memory, but also the fusion of storage and logic in the same hardware.

The potential benefits of in-memory computing, along with the high integration/storage density of a ReRAM crossbar array and its CMOS BEOL compatibility, are the driving force behind current efforts towards developing robust and viable solutions for computational ReRAM structures within the reach of today's technology. In this context, many logic families are being studied which exploit specific interconnection patterns of memristors that are compatible with the crossbar topology [3]. Some operate in a *stateful* way, using the resistance of memristor cells for logic input/output representation, whereas in other logic styles the resistance is used only for logic input data but the logic output is represented by voltage [4]. Selecting a logic style that does not imply conditional switching of memristors during information processing, could lead to higher reliability [5].

Given a specific logic primitive selected for in-memory computations, further considerations concern the adequate *memory driving circuitry*, co-designed to support both memory (read/write) and logic operations. It should guarantee a universal logic set and offer flexibility of data movement, required for multi-level (sequential) logic computations. Moreover, device variability is a reliability limiting factor, so its effects must be mitigated at circuit level with proper read/write methodologies that incorporate *testability* during normal circuit operation [6]. Furthermore, *ad-hoc synthesis tools* are required to bridge the gap between primitive logic gates and large-scale logic circuits. A synthesis process developed for ReRAM technology should be a combination of *logic* synthesis and *physical* synthesis. It should decompose an arbitrary logic function to a sequence of logic operations supported by the ReRAM driving circuit, and also compute the *spatio-temporal* properties of all the multi-level logic operations necessary for its execution in the physical ReRAM array. Solutions published so far in this field are almost exclusively oriented to *stateful* logic primitives [7].

In this direction, here we highlight important HW-SW co-design considerations and issues to be addressed towards the development of ReRAM-based computational memories. We support our claims through a case study concerning a synthesis flow (SW) for a *nonstateful* logic style [4] and the co-design of the underlying 1T1R crossbar array driving circuit (HW). We discuss certain topological and performance constraints that make imperative the HW-SW co-design, and also focus on the synthesis algorithm that computes the position in the crossbar array and the timing of the result of every logic operation.

Supported by Synopsys, Chile, by the Chilean grants FONDECYT Regular 1221747 and ANID-Basal FB0008, and by the Spanish MCIN/AEI/10.13039/501100011033 grant PID2019-103869RB-C33.

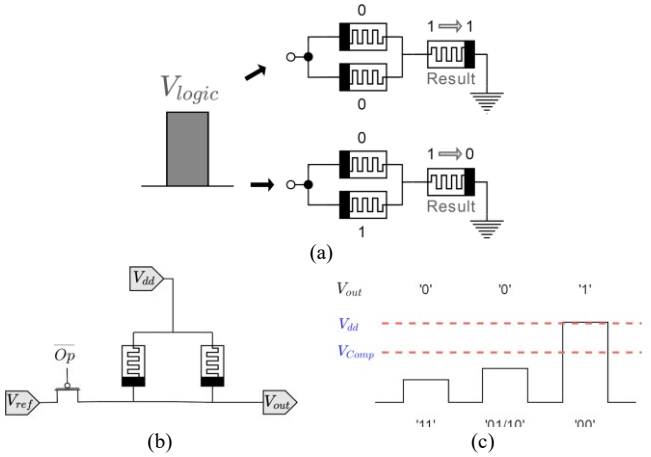

Fig. 1 Memristor-based logic primitives. (a) Schematic of a *stateful* logic style with three bipolar memristors [3]. (b) Schematic of a *nonstateful* ratioed-logic style for a NOR logic gate [4]. (c) Plot of the expected output voltage in (b) and the comparison for every input combination shown below.

Our synthesis approach minimizes the logic function latency by exploiting parallel execution of operations in a multi-level logic implementation, which are also carried out using the minimum required number of cross-point cells. Moreover, preliminary performance results prove the fast execution of the algorithm, owing to the *level-by-level mapping* of logic gates to the array cross-points which minimizes the potential exploration space.

II. Challenges in Computational ReRAMs

A. Basic Device Operation & Supported Logic Style

The devices most widely-studied and used in logic principles are bipolar. When a memristor is forward biased with a voltage pulse of amplitude higher than a V_{SET} threshold, then a SET process occurs and the resistance decreases towards a low resistive state (LRS or R_{ON}). On the contrary, when it is reverse-biased with a voltage pulse of amplitude higher than a $|V_{RESET}|$ threshold, then a RESET process occurs and the resistance increases towards a high resistive state (HRS or R_{OFF}). According to the read- out circuit used, the devices store a logic '0' with a HRS and a logic '1' with LRS (or vice versa). Bipolar memristors can be used for logic operations and a review of different logic styles can be found in [3]. The selected logic style should be independent of memristor device technology features and tolerant to variability. *Stateful* logic schemes normally use a series connection of memristors and functionally depend on the fact that, upon the application of a voltage, some memristor (previously initialized to a specific state) is subjected to a *conditional change* of its state which depends on the state of the memristors that store the input data. Fig. 1(a) shows an example representative of this concept. Note that, while in many works it has been assumed that such conditional switching of the "output" memristor is deterministic, under certain circumstances it becomes probabilistic [8]. Moreover, certain constraints apply for V_{logic} to guarantee that the operands´ state is unchanged, as desired.

However, solutions based on *nonstateful* logic primitives do not rely on probabilistic switching of memristors, and are less-prone to effects of variability. In such styles, computations are equivalent to modified memory read

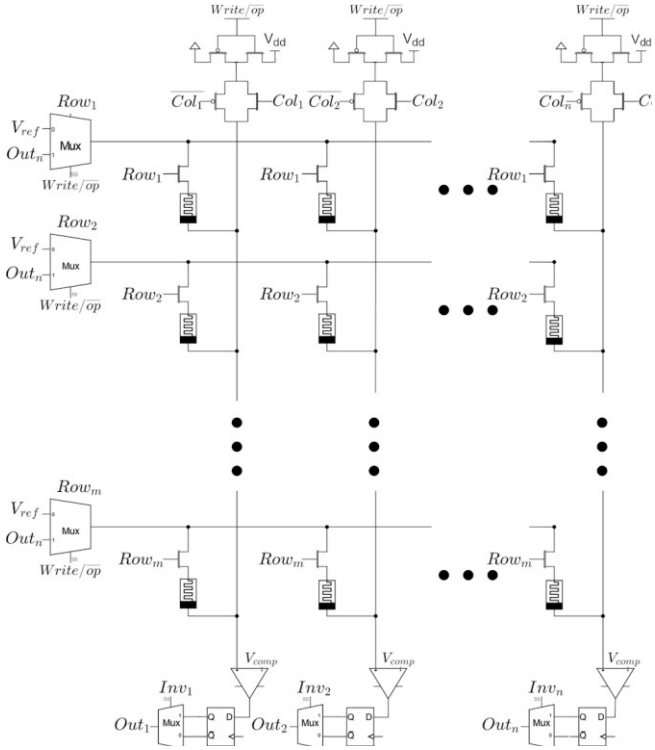

Fig. 2 1T1R crossbar with memristor-transistor cross-point cells. The schematic presents simplified driving circuitry for wordlines/bitlines, with the sensing scheme located at the lower part both for memory read and logic.

operations where logic output is available as voltage in the peripheral circuitry of the ReRAM array. Such read-based computing was applied in [9]. Logic output can be stored back to a memory element via a reliable memory write operation and no previous initialization of any output memristor is required. The circuit design shown in Fig. 1(b) supports such concept, whereas Fig. 1(c) shows the resulting voltage margins for reading logic '1' and '0' output. It corresponds to the logic design scheme proposed by Escudero *et al*. in [4]. The overall circuit is a voltage divider between a pull-down network of 2 parallel memristors and a PMOS transistor as resistive load tuned aprox. equal to the nominal LRS resistance of memristors. Such topology implements a *2*-input NOR gate. When at least one memristor is in LRS (logic '1'), the output voltage measured at the intermediate output node V_{out} drops significantly compared to when all devices are in HRS, thus output logic level is identified via comparison with V_{comp}. The same topology is applied to implement a NOT gate when only one input memristor is used.

B. Ad-hoc Read/Write Driver for Logic Operations

Adaptive write methods, multi-level sensing as well as complementary logic operations in the periphery and parallelization, are some issues to be taken into consideration in the design of the memory read/write drivers, which can require *asynchronous* circuit techniques. Moreover, design-for-test (DfT) techniques are required [6] to detect undefined state faults (USFs) during memory operation (not covered in this paper). In the context of our case study, Fig. 2 shows an overview of a 1T1R ReRAM array topology with a simplified driving and sensing circuitry to support *nonstateful* logic operations according to the style described in Fig. 1(b, c). Computation takes place in the periphery, while logic gate

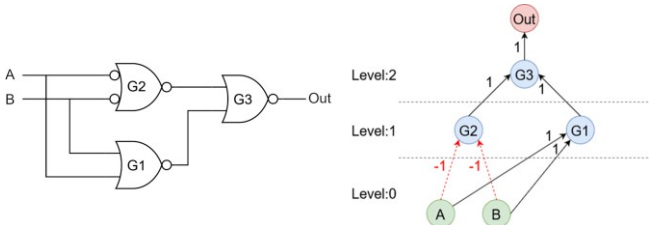

Fig. 3 A multi-level NOT/NOR logic implementation of a 2-input XOR function and the corresponding weighted directed graph representation.

inputs need to be aligned column-wise. The rows holding the input memristors are activated and row drivers apply V_{ref}, while target column drivers apply V_{dd}. Thus, by using a voltage comparator (and a reference voltage V_{comp}) followed by a register, the V_{out} levels for logic '0' and '1' are properly interpreted and captured. Moreover, through a MUX in the final stage, we have readily available the original and the complement of the memory/logic result. Hence, the circuit supports OR and NOR gates, whereas NOT and COPY operations are also possible in such extended universal logic set. If necessary, the output voltage can be converted to the corresponding resistive level once it is stored in a target memristor "*anywhere in the array*", via a reliable memory write operation. This way we achieve flexibility in the cross-point used to store the logic output. For instance, the destination cross-point can be aligned with another memristor to be used as input in posterior cascaded logic operations. This topology also supports parallel logic operations, exploited in the synthesis algorithm described in the following section.

C. *Ad-hoc Synthesis for ReRAM-Centric Computing System*

The synthesis will produce delay/area-efficient solutions, compatible with the co-designed peripheral circuitry of the ReRAM and the supported logic primitives. For a multi-level logic description of an arbitrary logic function, it will determine the array position of the inputs and the output of every logic gate in every level of computation, in the shortest possible time. Different optimizations, such as parallelization of operations per logic level, can lead to higher efficiency in delay or in the area occupied for computations. In the context of our case study, we present our early development steps towards a synthesis tool for the abovementioned ReRAM topology and logic scheme.

The necessary topology-dependent considerations are that the primary logic operation is the 2-input NOR and that the NOT gate is readily available in the periphery. Note that all input memristors of a NOR gate should be aligned column-wise. Logic gates in the same level are computed in parallel, thus their input memristors are aligned column-wise but in separate columns, because every crossbar output line (bitline) is the output node of a single logic gate. At the same time, all the cells acting as inputs in parallel gate computations need to be in the same wordlines (rows). Therefore, we use only 2-input NOR gates since it is not possible to execute in parallel NOR gates with unequal number of inputs. NOR/NOT based multi-level logic gate descriptions are mapped to a *weighted directed graph* where gates correspond to graph vertices and interconnections to graph edges, as shown in Fig. 3 for the XOR function. The edges with negative weight in Fig. 3 indicate complementary signals. The graph representation indicates existing dependencies of logic gates and the algorithm identifies the gates belonging to every logic level, whose output values can be computed simultaneously. So, the minimum number of steps required to compute a logic function *depends on the number of gate levels*. However, the exact number of system cycles depends on limitations imposed by the array topology; e.g. copying the same logic gate output to different array columns cannot be performed in a single step. With the NOT gate available in the periphery the required logic levels decrease, which has a positive impact to the total latency of computations. This said, the objective of logic synthesis is to find a correspondence between graph vertices and cross-points where the data are stored.

III. Synthesis Implementation & Performance Details

Figure 4 summarizes the synthesis flow, which was inspired on work in [7], [10], and is described below:

1) Netlist processing: We use the ABC tool [11] to generate the gate netlist and convert any logic function to equivalent expressions using NOR/NOT gates.

2) Graph mapping: Based on the ABC output netlist, the next stage creates the weighted directed graph whose vertices have the information that allows mapping to the crossbar:

- **Level**: the level of a particular vertex in the graph.

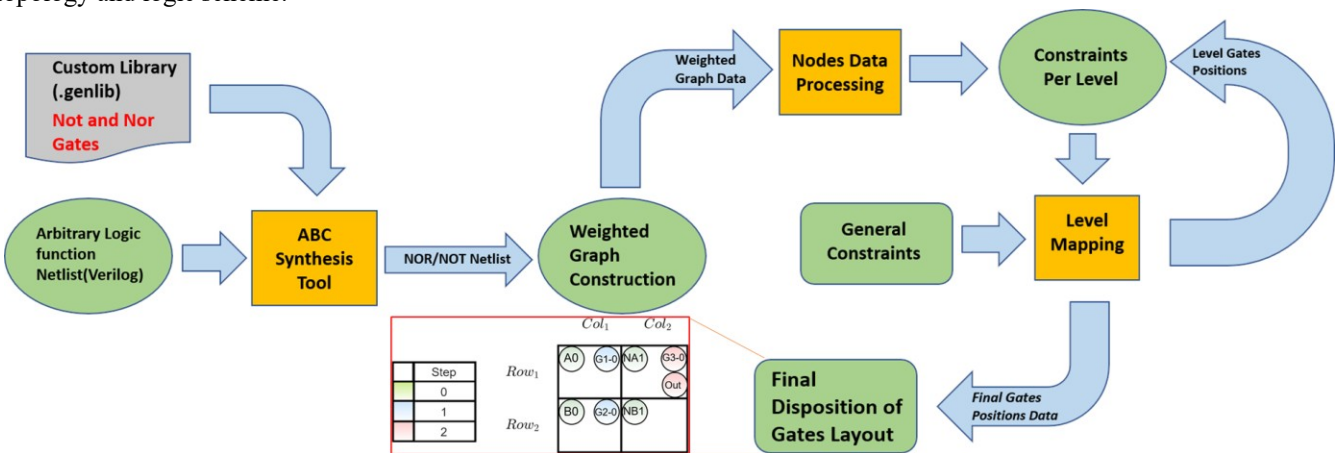

Fig. 4 Parts composing the developed synthesis flow, using the ABC tool and the Z3 solver [12]. The inset shows a graphical representation of the post-processed information given by the synthesis tool for the circuit in Fig. 3(b). The execution occupied a 2×2 crossbar. The layout shows spatio-temporal information for the data stored in all moments in the four cross-points. Level 0 is used to indicate the location of initial input values.

- **Input vertices**: vertices whose output value is input to a particular vertex.
- **Destination vertices**: vertices in subsequent levels to which the output of a particular vertex acts as input.
- **Edge weights**: weight applied to the output of a vertex for each destination vertex to which it acts as input.
- **Max. Level**: the maximum level with a vertex to which the output of a particular vertex acts as an input.

3) Mapping to crossbar array: The last stage consists in the mapping of the graph to the array cross-points, in every level. Essentially, we seek the position of every logic gate in the crossbar. To this end, we use the Z3 Theorem Prover [12] to satisfy the constraints associated with the memory topology and with the used logic style. Z3 finds the gate locations using the *level-wise mapping* of the graph vertices, so that all vertices belonging to the same level can be computed in parallel. Unlike other approaches in the literature, the proposed algorithm realizes the mapping in an iterative manner individually for every level to minimize the exploration space. The aforementioned constraints include:

- All memristors acting as inputs for a particular logic gate, should be *aligned in the same crossbar column*.
- Vertices should be mapped in *valid array positions*.
- For the parallel computation of vertices whose outputs act as inputs to the same gate in the next level, they both have to be mapped to the same column.
- Vertices acting as inputs to different gates should be mapped to different columns, but always in the same rows (wordlines) to be activated for logic operations.

Processing of the information delivered by Z3 results in a layout (see Fig. 4) for the position where the gate output values will be stored in every step. The outcome of the synthesis consists in *spatio-temporal information* for every graph vertex. We compute the logic value to be mapped in different levels of computation. When no requirement is placed to keep the input data, their cells can be overwritten to minimize to total cross-points used. We evaluated the performance of the synthesis flow using a subset of combinational benchmarks from LGynth93 suite [13]. Table I presents the number of inputs and outputs for every benchmark, the number of gate levels in the weighted directed graph (excluding Level 0), the minimum crossbar dimensions for which a solution was found by Z3, and the total time required for the synthesis in a PC with Intel Core i7-4510U CPU @ 2.00GHz and 16GB of RAM while Z3 was configured to find a potentially sub-optimal solution that satisfied the given area requirements. We compared our preliminary results with results from the Simple MAGIC (SM) synthesis flow, for the same input netlists given by the ABC tool. SM was used as configured in [7] for an exhaustive space-exploration to minimize latency. Footnote of Table I comments on time required by SM for some benchmarks. Performance on synthesis time and area requirement leave a positive impression for the potential impact of our work in this

TABLE I. SYNTHESIS RESULTS

Benchmark	Number of Inputs	Number of Outputs	Gate Levels	Crossbar Dimensions	Sintesis time (s)	Used cross-points	
						This work	*Simple MAGIC*
C17	5	2	3	2×4	0,841515	8	-
Parity	16	1	8	2×16	1,308009	32	240
x2	10	7	5	2×23	1,238206	46	168
misex1	8	7	5	2×21	1,319345	42	294
majority	5	1	4	2×6	0,7549	12	-
clip	8	5	9	2×129	27,274695	258	444
5xp1	7	10	7	2×59	3,280317	118	315
newtag	8	1	5	2×7	0,847877	14	-
CLPL	11	5	10	2×10	0,893361	20	-
t	5	2	3	2×4	0,811575	8	-
cm162a	14	5	8	2×18	1,18506	36	186
cm150a	21	1	9	2×31	1,277649	62	189

* Simple MAGIC synthesis took 2,3s for *C17*, 74,8s for *majority*, and 76,9s for *CLPL*.

in this field, whereas logic latency comparison requires further analysis of architectural aspects, not yet fully explored, as well as case-specific details.

IV. CONCLUSIONS & FUTURE WORK

Preliminary results for a first prototype of the proposed toolchain, demonstrate how parallel gate execution and *level-wise mapping optimizations* can lead to fast execution, and to more time-efficient multi-level function implementations, not considered elsewhere so far. Time-efficiency of the synthesis execution showed promising results owing to level-wise optimizations; a major advantage for its practical utilization.

REFERENCES

[1] C. Li, *et al*., "In-Memory Computing with Memristor Arrays," *in Proc. IEEE Int, Memory Workshop* (IMW), Kyoto, Japan, 13-16 May, 2018

[2] A. Sebastian, *et al*., "Temporal correlation detection using computational phase-change memory," *Nat. Comm*., vol. 8, no. 1115, 2017

[3] I.Vourkas and G. Ch. Sirakoulis, "Emerging memristor-based logic circuit design approaches: A review," *IEEE Circuits Syst. Mag*., vol. 16, no. 3, pp. 15–30, Third Quarter 2016

[4] M. Escudero, I. Vourkas, A. Rubio, and F. Moll, "Memristive Logic in Crossbar Memory Arrays: Variability-Aware Design for Higher Reliability", *IEEE Trans. Nanotechnol*., vol. 18, pp. 635-646, 2019

[5] X.Zhu, *et al*., "Implication of unsafe writing on the MAGIC NOR gate," *Microelectronics Jour*., vol. 103, pp. 104866, 2020

[6] E. Kondratyu, *et al*., "Automated Testing Algorithm for the Improvement of 1T1R ReRAM Endurance," *IEEE Trans. Electron Devices*, vol.68, no. 10, pp. 4891-4896, 2021

[7] R. Ben Hur, N. Wald, N. Talati, and S. Kvatinsky, "Simple magic: Synthesis and in-memory Mapping of logic execution for memristor-aided logic," 2017 *IEEE/ACM International Conference on Computer-Aided Design* (*ICCAD*), Irvine, CA, USA, Nov. 13-16

[8] R. Naous, *et al*., "Theory and experimental verification of configurable computing with stochastic memristors," *Sci. Rep*., vol. 11, no. 4218, 2021

[9] L. Xie *et al*., "Scouting logic: A novel memristor-based logic design for resistive computing," *in Proc. IEEE Comput. Soc. Annu. Symp. VLSI*, Bochum, Germany, Jul. 2017, pp. 176–181

[10] Z. Zhu, *et al*., "A General Logic Synthesis Framework for Memristor-based Logic Design," 2019 *IEEE/ACM International Conference on Computer-Aided Design* (*ICCAD*), Westminster, CO, USA, Nov. 4-7

[11] A. Mishchenko, "Abc: A system for sequential synthesis and verification," Accessed: May, 2020. [Online]. Available: http://www.eecs.berkeley.edu/~alanmi/abc/.

[12] N. Bjorner, "The z3 theorem prover," Accessed: May, 2020. [Online]. Available: https://github.com/Z3Prover/z3/graphs/contributors

[13] K. McElvain, "IWLS'93 Benchmark Set: Version 4.0," Distributed as a part of IWLS'93 benchmark set, May 1993